Blinker 5.0 Online-Referenz

Wenn Sie als Neuling im Umgang mit Blinker Probleme haben, empfehlen wir Ihnen, im Kapitel ‘Schnelleinstieg für verschiedene Compiler’ den Abschnitt, der Ihren Compiler behandelt, zu lesen und nach den dortigen detaillierten Anleitungen zum Erstellen, Linken und Starten eines Testprogramms vorzugehen, bevor Sie den technischen Support anrufen. In diesem Fall sollten Sie unbedingt die Informationen bereithalten, die im Abschnitt ‘Erforderliche Informationen für technischen Support’ weiter unten aufgelistet sind.

Im ersten Kapitel der ‘Online Dokumentation’ weiter unten, ‘Neue Leistungsmerkmale in 5.0’, sind Querverbindungen zu den Abschnitten enthalten, in welchen die in dieser Blinker-Version neuen Leistungen beschrieben werden. Wir bitten Sie auch, diese Abschnitte zu studieren, bevor Sie den technischen Support anrufen.

Abschnitte dieser Referenz

Schnelleinstieg für verschiedene Compiler

Erforderliche Informationen für technischen Support

Ausführliche Befehlsreferenz

Funktions-Kurzübersicht

Fehler und Fehlermeldungen

Die Umgebungsvariable BLINKER

Analyse von Allgemeinen Schutzverletzungen (GPFs)

Online-Dokumentation

Neue Leistungsmerkmale in 5.0

Einführung

Schnell-Einstieg

DOS-Extender- / Dual Mode-Programme

Windows-Programme

Entwicklung unter CA-Clipper

Real Mode-Programme

Allgemeine Debugger-Unterstützung

Spezielle Features

Link-Skriptbefehle

Blinker-Funktionen zur Programmierung

Anhänge

Blinker als DOS-Programm unter Windows

Siehe Eintrag zu:

Blinker unter Windows ausführen

Neue Leistungsmerkmale in 5.0

Dies sind die wesentlichen neuen Leistungen, die mit dieser neuen Blinker-Version eingeführt werden:

Allgemein

Deutlich schnellere Komprimierungsgeschwindigkeit und verbessertes Komprimierungsverhältnis für DOS-, DOS-Extenderprogramme und für 16-Bit Windowsprogramme.

Die Möglichkeit, ein DOS-Extenderprogramm als Stub für ein 16- oder 32-Bit Windowsprogramm einzusetzen.

Volle Jahr-2000-Fähigkeit.

Erforderliche Informationen für technischen Support

Neue Unterstützung für Compiler und Bibliotheken

Alaska Xbase++

Zinc Application Framework für DOS

Protected Mode Bibliotheken für die neuesten Versionen von Borland C/C++, Symantec C/C++ und WATCOM C/C++.

Volle Unterstützung für Lahey 32-Bit FORTRAN inclusive CodeView-Unterstützung und linken von 32-Bit DOS-Extenderprogrammen mit dem PharLap 32-Bit DOS-Extender, wobei typischerweise signifikant kleinere EXE-Dateien erstellt werden, wenn keine Komprimierung verwendet wird.

CA-Clipper

DOS-Extenderprogramme, die in CA-Clipper 5.2d/e und 5.3b geschrieben sind, haben nun einen ERHEBLICH vergrößerten Eval-Stack und Raum für statische Variablen, wodurch der Fehler 667 (Eval-Stack-Fehler) eliminiert sein dürfte.

Verbessertes inkrementelles Linken von CA-Clipper-Programmen für DOS Extender und Windows.

Korrigiertes MPAR.OBJ aus der CLIPPER.LIB, um gelegentliche Ladezeitfehler in CA-Clipper Windowsprogrammen zu beseitigen.

Die Möglichkeit, EXE-Komprimierung bei sehr großen Clipper-Symboltabellen in einem Windowsprogramm zu nutzen.

Erzeugen einer CA-Clipper Windows Anwendung

CA-Clippers DGROUP maximieren mit //INFO

Weitere neue Leistungen und Verbesserungen

Verbesserte Unterstützung für die neuesten Versionen von SoftIce/95 und SoftIce/NT.

Deutliche Vergrößerung der maximalen Größe einer .EXE- und .DLL-Ausgabedatei.

Verbesserte Skriptdatei für MS Link und Tlink und verbessertes Abarbeiten von Parametern.

Blinker als DOS-Programm unter Windows

Einführung

Übersicht der Blinker Merkmale

Los geht’s

Installation

Einstellung der Umgebung

DOS Version des Blinkers unter Windows ausführen

Registrierung und Technische Unterstützung

Lizenzabkommen

Übersicht der Blinker Merkmale

Blinker ist ein preisgekrönter dynamischer Overlay-Linker, ein 286-er DOS-Extender und ein Windows-Linker. Er wurde für Programmierer erstellt, die ein schnelles und leistungsfähiges Entwicklungswerkzeug benötigen. Blinker ist der einzige Linker, über den ein Programm erzeugt werden kann, das sowohl im Real als auch im Protected Mode von DOS läuft. Außerdem erzeugt nur Blinker direkt ausführbare komprimierte .EXE und .DLL Dateien für Windows- und DOS-Extended-Programme. Von den vielen Merkmalen sind besonders hervorzuheben:

DOS-Extender

16- und 32-Bit-Windows-Linker

.EXE und .DLL Dateikomprimierung

Dual Mode-Programme

Dynamischer Overlay Linker für DOS

Allgemeines

CA-Clipper spezifische Zusätze

Übersicht der Blinker Merkmale

DOS-Extender

Mit dem DOS-Extender Blinker gelinkte Programme haben ab sofort Zugriff für Code wie auch für Daten auf bis zu 16 MB verfügbaren Speicher. Blinker beinhaltet ein Compiler-unabhängiges VM-System, durch das ein transparenter Zugriff auf bis zu 64 MB virtuellen Speicher ermöglicht wird.

Die mit einem DOS-Extender erzeugten Programme sind zu allen DOS-Netzwerken, speicherresidenten Programmen etc. voll kompatibel und können in allen möglichen Umgebungen ausgeführt werden, da der Extender alle wichtigen Protected Mode-Schnittstellen unterstützt:

- DPMI, zum Betrieb in einer DOS-Box unter Windows und OS/2,
- VCPI, zum Betrieb unter Speichermanagern die das DPMI nicht unterstützen, wie zum Beispiel ältere Versionen des QEMM und 386MAX,
- XMS, für einfache, 80286-basierende Speichermanager.

Das Programm wird in den erweiterten Speicher geladen und von dort gestartet, wobei der Rest des verfügbaren Speichers und 64 MB virtueller Speicher für seine Daten zur Verfügung gestellt werden. Das bedeutet, es muß kein Code mehr geswappt werden, auch keine Daten, solange ausreichend Erweiterungsspeicher zur Verfügung steht.

Da solch ein Programm in den Erweiterungsspeicher geladen wird, verringern sich auch die DOS-Speicheranforderungen enorm. Die meisten Programme sind nun in der Lage, mit weniger als 200 KB verfügbarem konventionellen DOS-Speicher auszukommen, wodurch sie praktisch selbst auf besonders stark belasteten Netzwerkstationen ausgeführt werden können.

DOS-Extender Programme können nun auch DLLs (Dynamic Link Libraries) verwenden, um Code mehreren Applikationen gemeinsam zur Verfügung zu stellen, wobei Programmteile ganz einfach ersetzt werden können.

Übersicht der Blinker Merkmale

16- und 32-Bit-Windows-Linker

Blinker erzeugt 16- und 32-Bit-Windows-kompatible Programme für Compiler, die eine Entwicklung für Windows unterstützen und Microsoft-kompatible OBJ-Dateien generieren. Dies sind alle C/C++ Compiler, Assembler, Microsoft Pascal und FORTRAN, und CA-Clipper 5.x (nur im 16-Bit-Modus und unter Verwendung von Clip4Win oder FiveWin).

Es sind keine Änderungen der Standardmoduldefinitionsdateien erforderlich.

Ein neuer Befehl für Moduldefinitionsdateien

```
RC [<Resourcedateiname>]
```

ermöglicht Blinker das Einbinden einer Windows Resourcedatei in das EXE-Programm zur Linkzeit, wobei der letzte Durchgang eines Resource-Compilers nicht mehr benötigt wird.

BLINKER.EXE kann unter Windows oder unter DOS betrieben werden - er erkennt automatisch seine Umgebung, in der er betrieben wird.

Übersicht der Blinker Merkmale

.EXE und .DLL Dateikomprimierung

Durch den einfachen Zusatz eines Kommandos zur Link-Skriptdatei kann Blinker Windows-, DOS- und DOS-Extenderprogramme um 50% oder mehr komprimieren. Diese komprimierten Programme können in dieser Form gespeichert und vertrieben werden.

Die Verwendung von Blinker's EXE- und DLL-Komprimierung für den letzten Linklauf vor dem Versand bietet den Vorzug, die Versandkosten zu minimieren, (so wie es bisher durch traditionelle Komprimierungsprogramme erzielt wurde,) bietet im Unterschied dazu jedoch den bequemen Vorteil auf externe Dekomprimierungsprogramme bei der Installation verzichten zu können. Der Anwender merkt garnicht, daß das Programm komprimiert ist.

Die Kompression optimiert auch die Netzleistung in LAN, WAN und Internet durch verringerten Datenverkehr, und sie schützt die Software vor Dekompilierung.

Übersicht der Blinker Merkmale

Dual Mode-Programme

Die Möglichkeit, einfach durch Hinzufügen eines einzelnen Befehls in die Link-Skriptdatei Dual Mode-Programme zu erzeugen, stellt einen wesentlichen Durchbruch dar. Dual Mode-Programme erkennen bei ihrem Aufruf, ob genügend Systemresourcen zur Verfügung stehen, um das Programm im Protected Mode ablaufen zu lassen. Sind nicht genügend Ressourcen vorhanden, wird das gleiche Programm als normales Programm mit dynamischen Overlays ausgeführt.

Die EXE-Datei besteht aus nur einem Programm mit der Fähigkeit, in jedem Modus abzulaufen, und nicht aus zwei separaten Kopien, irgendwie miteinander verbunden. Der Programmzuwachs in der EXE-Datei gegenüber einem Single Mode-Programm ist geringfügig. Auch die Arbeitsgeschwindigkeit verringert sich vernachlässigbar, da ja das gesamte Programm in den Speicher geladen wird, wenn es im Protected Mode ausgeführt wird.

Damit der Dualmodus funktioniert und auch effektiv ist, muß sämtlicher Code so geschrieben sein, daß er sowohl im Real als auch im Protected Mode ausgeführt werden kann. Für Hochsprachen wie CA-Clipper sind keine Änderungen im Quelltext erforderlich, um ein Programm Protected Mode-fähig zu gestalten. Änderungen in C-, PASCAL- oder FORTRAN-Programmen sind eher die Ausnahme von der Regel. Diese Anforderung betrifft lediglich die C- und Assembler-Systemprogrammierer. Im schlimmsten Fall muß der Code eine Testroutine enthalten, durch die der Prozessor-Modus jedesmal bei der Ausführung des Code ermittelt wird, und danach in den geeigneten Code übergewechselt werden.

Übersicht der Blinker Merkmale

Dynamischer Overlay Linker für DOS

Blinker enthält langjährige Erfahrungen in der Erstellung kleiner, schneller und speichereffizienter .EXE-Dateien, die auf allen Rechnern mit allen Prozessoren unter allen Versionen von DOS ab Version 3.0 aufwärts laufen. Diese Programme sind in der Lage, optional jeglichen freien EMS-, XMS- und UMB-Speicher zu nutzen, um die Arbeitsgeschwindigkeit der Programme zu erhöhen und den Programmen mehr Speicher zur Verfügung zu stellen. Des weiteren wird das Speicher-Caching des Overlays über XMS oder EMS zur Erhöhung der Arbeitsgeschwindigkeit unterstützt.

Mit Blinker entfällt in allen Sprachen die Notwendigkeit der Erarbeitung von Overlay-Strukturen. Die Programmentwicklung wird vereinfacht, wobei diese Vereinfachung mit Funktionalität verknüpft wird. Der Programmcode kann in dynamischen Overlays abgelegt werden, um eine optimale Nutzung des Speichers zu erreichen, ohne daß komplexe Overlay-Strukturen angelegt werden müßten. Die unter DOS bestehende 640 KB-Speicherbegrenzung wird einfach und effektiv umgangen. Um dem Anfänger die Einarbeitung innerhalb weniger Minuten zu ermöglichen, wurden Beispiel-Linkdateien mitgeliefert.

Übersicht der Blinker Merkmale

Allgemeines

Linkgeschwindigkeit

Blinker ist der schnellste verfügbare Overlay-Linker. Er behält diesen Geschwindigkeitsvorteil sowohl für DOS-Extender- als auch für Windows-Programme bei. Im speziellen Falle des Linkens von Programmen mit dem DOS-Extender ist die Linkgeschwindigkeit gewöhnlich 5 - 10 Mal höher als die Geschwindigkeit bei Konkurrenzprodukten.

Zur Linkzeit läuft Blinker entweder als ein Windows- oder als ein DOS-Extended-Programm, je nachdem von wo aus er gestartet wurde und kann so allen verfügbaren Speicher des Systems verwenden. Allein durch die Zeiteinsparung in der Entwicklungsphase macht sich Blinker bereits innerhalb weniger Tage bezahlt.

Speicher-Swapper

Die integrierte Funktion zum Auslagern von Speicher ermöglicht es, ein Programm zu unterbrechen, um andere Programme, wie zum Beispiel eine Textverarbeitung, ein Sicherungsprogramm, aufzurufen oder gar zum DOS-Prompt zu wechseln.

Der Hauptunterschied zwischen dieser Funktion und der C-Funktion ‘EXEC()’, dem BASIC-Befehl ‘SHELL’ oder dem CA-Clipper-Befehl ‘RUN’ ist folgender: Fast der gesamte Speicher, der durch den übergeordneten Prozeß belegt wurde, wird nun dem untergeordneten Prozeß (Child) verfügbar gemacht.

Dies wird erreicht, indem der gesamte oder nur Teile des übergeordneten Prozesses auf die Festplatte oder in EMS/XMS gesichert werden und lediglich ein kleiner residenter Kern im Speicher verbleibt. Dann wird der untergeordnete Prozeß ausgeführt, danach das Abbild des übergeordneten Prozesses wiederhergestellt und dessen Ausführung fortgesetzt. Die Größe des residenten Kerns variiert zwischen 0 und 11 KB.

Debugger Unterstützung

Blinker unterstützt im vollen Umfang sog. ‘Source-Level-Debugging’ bei Windows-, DOS Real Mode- und DOS Protected Mode-Programmen für alle Compiler mit den Soft-Ice/W- und Periscope-Debuggern. CodeView und andere CodeView-kompatible Debugger werden voll für Windows und den DOS Real Mode unterstützt.

Demo Versionen

Mit Blinker ist man in der Lage, Demo-Versionen zu erstellen, ohne den eigentlichen Programm-Quelltext ändern zu müssen. Die Benutzung der Demo-Version kann eingeschränkt werden: über die Angabe der Minuten, wie lange diese Demo-Version laufen soll bis hin zur Angabe eines Verfall-Datums, nach denen bzw. nach dem das Programm jeweils beendet oder abgebrochen wird.

Seriennummern-Zusatz

Blinker ermöglicht es dem Entwickler, einen Text (z.B. eine Seriennummer oder den Firmennamen) als eine Form von Kopierschutz im ausführbaren Programm verschlüsselt abzulegen.

Industriestandard-Dateiformate

Die Link-Skriptdateien besitzen das Industriestandard-Format FREEFORMAT, wodurch einfache, klare und leistungsfähige Befehle zur Angabe der Programmteile wie auch der Kommentare gegeben sind. Allen Blinker-spezifischen Befehlen wurde zwecks einfacher Identifikation das Schlüsselword BLINKER vorangestellt.

Die Moduldefinitionsdateien für Windows-Programme sind Windows-Standarddateien (.DEF).

Die mit dem DOS-Extender erzeugten .EXE-Dateien besitzen das standardmäßige Windows-Dateiformat für segmentierte .EXE-Dateien. Programme mit diesem Format können effizient gespeichert werden, wodurch kurze Ladezeiten und ein Maximum an Kompatibilität zu Kopierschutz- und Anti-Virus-Programmen erreicht werden.

Ein Entwicklungs-Tool für viele Programmierumgebungen

Blinker ist ein hochentwickeltes Entwicklungs-Tool für alles, was für die PC-Programmierung erforderlich ist. Er unterstützt die wichtigsten Programmiersprachen in den wichtigsten PC-Umgebungen und hilft, die Software-Kosten, Programmierer-Schulungen und nachfolgende Supportprobleme wesentlich zu verringern.

Die Entwicklung, Dokumentation und technische Beratung erfolgt insgesamt in einem Haus. Daher gibt es keine Abhängigkeit von anderen Firmen in Bezug auf Updates, technische Informationen oder gar Bug-Fixes. Blinker besitzt einen exzellenten technischen Support in der ganzen Welt und läßt die Kunden auch nach dem Kauf nicht im Stich.

Keine Laufzeit-Lizenzgebühren

Blinkinc verlangt für keine der mit Blinker erstellten Programme Laufzeit-Lizenzgebühren. Die einzige Forderung besteht darin, daß jeder Programmierer mit seiner eigenen Kopie von Blinker arbeitet. Weiterhin wird nur für einen registrierten Benutzer pro Produkt Support geleistet.

Übersicht der Blinker Merkmale

CA-Clipper spezifische Zusätze

Inkrementelles Linken von DOS-, DOS-Extender- und Windows-Programmen

Blinker erreicht beim inkrementellen Linken für CA-Clipper Sommer ‘87 und auch für 5.x eine phänomenale Linkgeschwindigkeit. Dieses Produkt ist das einzige Programm, mit dem CA-Clipper 5.x-Programme für den Einsatz im Protected Mode oder unter Windows inkrementell gelinkt werden können. Bei einem erneuten Linklauf werden lediglich die Teile der Applikation neu gelinkt, die sich seit dem letzten Linkvorgang geändert haben. Linkzeiten unter einer Sekunde selbst bei größten Applikationen können so erzielt werden.

Komprimierung der Symboltabelle

Nicht benötigte Symbole werden durch diese Komprimierung aus der CA-Clipper Symboltabelle entfernt (Sommer ‘87 und 5.x). Dadurch reduziert sich in beiden Fällen der Speicherbedarf und die Größe der .EXE-Datei.

Dynamisches ‘Paging-System’

Blinker unterstützt dieses System bei Clipper 5.x, um die Ausführungsgeschwindigkeit der Anwendung zu optimieren. Außerdem wird der Speicherbedarf für Overlay-Code (CA-Clipper 5.x) reduziert.

Einbrennen der Umgebung

Der Entwickler hat die Möglichkeit, den Namen der CLIPPER-Umgebungsvariablen, ebenso wie die Vorgabewerte dieser Variablen zu verändern und Voreinstellungen für diese Variablen in die ausführbare Datei ‘einzubrennen’.

Los geht's

Ihr Blinker-Paket besteht aus der Distributionsdiskette und dem Blinker-Benutzerhandbuch (dem vorliegenden Handbuch). Die Distributionsdiskette enthält alle zur Arbeit mit Blinker und allen unterstützen Compilern notwendigen Dateien und eine umfangreiche Online-Referenz der Befehle, Funktionen und Fehlermeldungen .

Bevor Sie Ihre Arbeit mit Blinker beginnen, fertigen Sie sich bitte eine Arbeitskopie der Distributionsdiskette an und verwahren Sie die Originaldiskette an einem sicheren Platz. Verwenden Sie die Originaldiskette als Sicherung und installieren Sie Blinker von der eben angefertigten Arbeitskopie.

Die LIESDAS.WRI-Datei

System-Anforderungen

Installation

Einstellung der Umgebung

MS Link und TLINK Link-Skriptdateien

Los geht's

Die LIESDAS.WRI-Datei

Es ist sehr wichtig, daß Sie sich Zeit nehmen, die LIESDAS.WRI-Datei (unter MS Windows Write) oder README.TXT (DOS Textdatei) auf der Diskette zu lesen, bevor Sie mit Blinker zu arbeiten beginnen. Diese Datei befindet sich im Blinker-Verzeichnis. Sie enthält Informationen, die möglicherweise erst nach Drucklegung dieses Handbuches vorlagen.

Informationen in LIESDAS.WRI oder in README.TXT haben Priorität vor den entsprechenden Informationen im Handbuch.

Die Datei PACKING.LST zeigt alle Dateien auf der Distributionsdiskette zusammen mit einer kurzen Beschreibung jeder Datei.

Los geht's

System-Anforderungen

Mindestanforderung für den Start von Blinker: Ein IBM AT, PS/2 oder kompatibler Rechner mit 286-er Prozessor oder höher, DOS-Version 3.0 oder höher, ein 3,5 Zoll Diskettenlaufwerk und eine Festplatte. Auf der Festplatte werden nach der Installation aller Blinker-Dateien etwa 3 MB Speicherplatz benötigt.

Blinker kann als eigenes Windows-Programm betrieben werden - hierzu wird ein Icon erzeugt - oder in normaler Weise auf Kommandoebene mit der Übergabe eines Link-Skriptdateinamens. Er kann wahlweise auf der DOS-Befehlszeile oder in einer DOS-Box unter Windows 3.0 oder höher, Windows

95/98, Windows NT, OS/2 1.1 oder höher und VM/386 betrieben werden.

Blinker erzeugt DOS-Programme und 16-Bit/32-Bit-Windows-Programme - falls vom Compiler unterstützt - für folgende Compiler:

Borland C/C++
CA-Clipper Summer '87 (2:00am Version und höher)
CA-Clipper 5.x
FORCE 2.x
Microsoft Assembler 4.0, 5.x, 6.x
Microsoft BASIC PDS 7.1
Microsoft C 5.0, 5.1, 6.0, 7.0
Microsoft FORTRAN 5.0, 5.1
Microsoft Pascal 4.0
Microsoft QuickBASIC 4.5 (einige Einschränkungen)
Microsoft Visual BASIC für DOS
Microsoft Visual C/C++
RM/FORTRAN
Stony Brook Modula-2 und Pascal
Symantec C++
Turbo Assembler
WATCOM C/C++ 8.0 und höher

sowie für die meisten Compiler, die das sog. ‘Large Model’ erzeugen und die in der Lage sind, Objektmodule im Microsoft / Intel Format zu generieren.

Der Speicher-Manager von Blinker unterstützt diejenigen Extended- und Expanded-Memory Speichertreiber, die sich an folgende Standards halten:

- EMS Version 3.2 und höher
- XMS Version 2.0 und höher
- VCPI
- DPMI

Los geht's

Installation

Bevor Sie nun den Blinker installieren, nehmen Sie sich bitte einige Minuten Zeit, um die beiliegende Registrierungskarte auszufüllen und zurückzusenden. Es ist wichtig, daß Sie dies tun, denn nur so stehen Ihnen 90 Tage kostenloser Support zu. Außerdem können Sie nur dann weiteren Support und zukünftige Upgrades in Anspruch nehmen.

Um den Blinker zu installieren, legen Sie nun bitte die Diskette in Ihr Laufwerk ein und geben Sie auf dem DOS-Prompt ein:

```
X:\INSTALL
```

wobei X: die Bezeichnung des Laufwerks darstellt, in dem sich die Distributionsdiskette befindet. Das Installationsprogramm wird Sie nun fragen, auf welches Laufwerk und in welches Verzeichnis Sie den Blinker installieren wollen und dies dann durchführen.

Installation der CA-Clipper Link Skript-Dateien

Diese Version von Blinker enthält Skript-Dateien, die, wenn sie eingesetzt werden, Teile von Bibliotheksdateien mit in den Overlay-Bereich legen. Diese Dateien werden in das CLP\LNK Unterverzeichnis des Blinker Hauptverzeichnisses installiert.

Wenn ein Verweis von Ihrer Skript-Datei auf eine andere Skript-Datei erfolgt, sucht Blinker in der nachstehenden Reihenfolge nach dieser Datei:

- im explizit gesetzten Pfad - falls angegeben,
- im aktuellen Verzeichnis,
- in den Verzeichnissen, die mit der Umgebungsvariablen LIB angegeben wurden.,

Wir empfehlen daher, daß Sie entweder:

Ihr Blinker-Verzeichnis in die Umgebungsvariabel LIB mit aufnehmen:

```
SET LIB=C:\CLIPPER5\LIB;C:\BLINKER\CLP\LNK
```

oder

die Skript-Dateien in das Verzeichnis kopieren, auf das die Umgebungsvariable LIB verweist:

(angenommen, LIB zeigt auf C:\CLIPPER5\LIB)

```
COPY C:\BLINKER\CLP\LNK\*.LNK C:\CLIPPER5\LIB <enter>
```

Die CA-Clipper 5.3 Workbench Link-Templates für Real-Mode, Dual- Mode und Extended-Mmode Programme befinden sich im Unterverzeichnis CLP\TPL als Dateien BLIRMD.TPL, BLIDUA.TPL und BLIEXT.TPL . Beachten Sie, daß incrementelles Linken defaultmäßig auf ON steht, wenn Sie diese Templates verwenden.

Los geht's

Einstellung der Umgebung

Die Variable PATH

Die Umgebungsvariable PATH sollte um das Verzeichnis erweitert werden, in das BLINKER.COM, BLINKER.DLL und BLINKER.EXE installiert wurden. Das Installationsprogramm erweitert die PATH-Variable automatisch mit dem Pfad zum BIN Unterverzeichnis des Blinkers.

Wenn Sie die Blinkerdateien in ein anderes Verzeichnis kopieren oder verschieben, denken Sie daran, die PATH-Variable entsprechend anzupassen, und achten Sie darauf, die obigen drei Dateien alle zu kopieren oder zu verschieben und sie alle im gleichen Verzeichnis zu halten.

Blinker verwendet die folgenden Umgebungsvariablen, um zu Link-Zeit Objekt-Module und Bibliotheken zu finden:

Die Variable OBJ

Nehmen wir einmal an, die in der Skript-Datei angegebenen .OBJ-Dateien liegen nicht im aktuellen Verzeichnis und es wurde kein Pfad bei dem FILE-Befehl angegeben. In diesem Fall kann man mit Hilfe der Umgebungsvariablen OBJ eine Liste der Pfade angeben, in denen Blinker die erforderlichen .OBJ-Dateien findet. Es ist sinnvoll, dabei das Verzeichnis anzugeben, in dem sich alle die .OBJ-Dateien befinden, die der Compiler regelmäßig benötigt. Sinnvoll wäre ebenfalls die Angabe des Verzeichnisses, in dem sich Bibliotheken und .OBJ-Dateien von Fremdanbietern befinden. Natürlich können Sie auch beide Pfadangaben setzen.

Die Variable LIB

Diese Variable enthält wiederum eine Liste von Pfadangaben, durch die Blinker mitgeteilt wird, in welchen Verzeichnissen er nach Bibliotheken suchen soll, die sich nicht im aktuellen Verzeichnis befinden. Blinker verwendet diese Variable auch, um Skript-Dateien zu suchen, die sich nicht im aktuellen Verzeichnis befinden.

Die Variable TMP oder TEMP

Die Umgebungsvariable TMP oder TEMP kann gesetzt werden, um Blinker dahingehend zu informieren, wo er seine Temporärdateien ablegen soll, falls zu wenig Speicher für den Linklauf zur Verfügung steht. Auch der DOS-Extender von Blinker verwendet diese Variable, um festzulegen, wohin er seine virtuelle Speicher-'SWAP'-Datei ablegen soll, für den Fall, daß bei der Programmausführung zu wenig Hauptspeicher zur Verfügung steht.

Weitere Einzelheiten zu diesen Umgebungsvariablen und eine genaue Beschreibung der BLINKER Variablen zur Steuerung eines Programms während der Laufzeit finden Sie im Abschnitt 'Umgebungsvariablen' in Kapitel 2.

Los geht's

MS Link und TLINK Link-Skriptdateien

Blinker erzeugt eine Datei BLINKER.INI, die sich im gleichen Verzeichnis wie BLINKER.EXE befindet. Diese INI-Datei enthält globale Einstellungen zur Steuerung des Verhaltens von Blinker zur Linkzeit. Der

wichtigste Eintrag in dieser Datei ist die Zeile ‘LinkScript=’, die das Link-Skriptdateien-Format bezeichnet, das Blinker erwartet. Die Datei BLINKER.INI kann in das aktuelle Verzeichnis kopiert werden, wenn die globalen Einstellungen für ein spezielles Programm oder Projekt überschrieben werden sollen.

Die Voreinstellung von ‘LinkScript’ ist BLINKER, da die meisten Blinker-Eigenschaften nur über die leistungsstarken Freiformat-Link-Skriptdateien angewendet werden können.

Die Einstellung kann in MSLINK oder TLINK geändert werden (es wird nicht zwischen Groß- und Kleinbuchstaben unterschieden), wodurch alle Link-Skriptdateien im entsprechenden Format verarbeitet werden. Die meisten Optionen mit ‘/’ werden von diesen Linkern auch unterstützt, nicht-unterstützte Optionen werden einfach übergangen.

Um beispielsweise eine MS-LINK-Skriptdatei zu verwenden, muß die Datei BLINKER.INI folgenden Eintrag enthalten:

```
. . .
[General]
LinkScript=MSLINK
. . .
```

DOS version des Blinkers unter Windows ausführen

Blinker erkennt automatisch die Umgebung, in der er betrieben wird und kann sowohl als Windows- als auch als DOS-Programm ausgeführt werden. Um Blinker aus der DOS-Befehlszeile zu starten, geben Sie wie bei jedem anderen DOS-Programm einfach BLINKER ein. Hiermit wird BLINKER.EXE geladen und als DOS-Programm ausgeführt.

Unter Windows 95/98 und Windows NT werden Windows-Programme automatisch erkannt, daher läuft hier Blinker immer als Windows-Programm ab, selbst in einer DOS-Box. Aus diesem Grund wird das Programm BLINKER.COM mitgeliefert, das die Ausführung von Blinker von der DOS-Befehlszeile aus ermöglicht. Geben Sie auf dem DOS-Prompt BLINKER ohne Erweiterung ein, führt DOS das Programm BLINKER.COM statt BLINKER.EXE aus, da es .COM- vor .EXE-Dateien sucht.

Alternativ können Sie über das Programm BLICFG.EXE den Betriebssystemschalter im Programm BLINKER.EXE auf Nur-DOS setzen, um eine Ausführung als Windows-Programm ganz zu unterbinden. Durch BLICFG /XD setzen Sie das Betriebssystem für Blinker auf Nur-DOS, über BLICFG /XW wird der Schalter wieder auf die Standardeinstellung Windows zurückgesetzt, was ein Betreiben von Blinker als Windows- und DOS-Programm wieder ermöglicht.

Um Blinker als Windows-Programm auszuführen, legen Sie für ihn einfach ein Symbol oder eine Verknüpfung in Verbindung mit der entsprechenden Befehlszeile an und Blinker wird als Windows-Programm ausgeführt, sobald Sie das Symbol anwählen. Sie können das Fenster ‘Taste drücken’, das in der Windows-Version nach Beenden des Linkvorgangs angezeigt wird, abschalten, indem Sie Ihrer Link-Skriptdatei den Befehl BATCH hinzufügen.

Registrierung und Technische Unterstützung

Blinkinc bietet registrierten Anwendern kostenfreie technische Unterstützung in den USA und Kanada wie nachfolgend aufgeführt. Technische Unterstützung für andere Länder finden Sie weiter unten in diesem Kapitel.

Wichtige Hinweise

Bitte füllen Sie die Regstrierungskarte vollständig aus und schicken Sie sie uns, damit Sie berechtigt sind, unsere technische Beratung in Anspruch zu nehmen und Upgrade-Informationen zu erhalten.

Wir unterstützen einen Benutzer pro Lizenz, und zwar genau den, der auf der Registrierungskarte eingetragen ist. Wir wollen qualitativ hochwertige technische Unterstützung für registrierte Benutzer anbieten. Es ist daher nicht unsere Politik, gleichzeitig mehrere Benutzer einer Lizenz zu unterstützen. Wenn Sie möchten, daß wir den Namen der Person ändern, die berechtigt ist, den technischen Support in Anspruch zu nehmen, schreiben Sie bitte an die weiter unten stehende Adresse.

Wenn Sie uns wegen technischer Unterstützung kontaktieren, benötigen wir mindestens die im nächsten Abschnitt aufgezählten Informationen.

Erforderliche Informationen für technischen Support

Technischer Support per Email

Technischer Support in USA und Kanada

Europäische Kunden

Übersee

Anschrift in den USA

Registrierung und Technische Unterstützung

Erforderliche Informationen für technischen Support

Bitte versuchen Sie zunächst, eine Antwort im Handbuch, inder Online-Hilfe oder auf unserer Web Seite zu finden, bevor Sie mit uns Kontakt aufnehmen.

Überprüfen Sie bitte auch, ob das Problem mit einem anderen Linker nicht auftritt und versuchen Sie, es mit einem kleinen Testprogramm zu reproduzieren. Falls das Problem in einem kleinen Testprogramm nicht auftritt, isolieren Sie es wenigsten soweit, daß es leicht reproduzierbar ist.

Wenn Sie nun mit uns wegen technischer Unterstützung Kontakt aufnehmen, dann brauchen wir mindestens folgende Informationen:

- Ihre Blinker-Seriennummer und die Versionsnummer,
- den Typ und die Version des Betriebssystems, unter dem das Problem auftritt,

- den Namen und die Versionsnummer des Sprachcompilers, mit dem Sie arbeiten,
- die Namen und Versionsnummern aller Add-On-Produkte, die Sie verwenden,
- den Inhalt Ihrer Link-Skript-Datei und der MAP-Datei,
- was hat sich geändert seit dem Zeitpunkt, an dem das Programm zuletzt richtig funktioniert hat,
- die exakte Fehlernummer und/oder der exakte Fehlertext,
- eine Kurzbeschreibung Ihres Problems und ob er auf mehreren Maschinen oder Betriebssystemen auftritt.

Falls der Fehler ein GPF ist, benötigen wir zusätlich den Text der GPF Meldung und das Symbol, das den Fehler verursacht hat. Dies ist unter ‘Meldung eines GPFs an den technischen Support’ in der Sektion ‘Analyse eines General Protection Faults (GPF) ’ der Online Hilfe beschrieben.

Meldung eines GPF an den technischen Support

Registrierung und Technische Unterstützung

Technischer Support per Email

Blinkinc USA bietet registrierten Anwendern der aktuellen Blinkerversion technische Unterstützung über seine Webseite im Internet. Die Webseite www.blinkinc.com enthält die neuesten Blinker Updates, technische Supporthinweise und andere interessante Dateien für Entwickler.

Bitte versuchen Sie zuerst, eine Antwort im Handbuch, in der Online-Hilfe oder auf unserer Webseite zu finden, bevor Sie mit uns Kontakt aufnehmen. Wenn Sie dann immer noch Schwierigkeiten haben, können Sie sich mit den Fragen, Problemen oder Vorschlägen an blinkersupport@blinkinc.com wenden. Vorschläge für Verbesserungen sind jederzeit willkommen, und Emails werden normalerweise innerhalb von 48 Stunden beantwortet.

Registrierung und Technische Unterstützung

Technischer Support in USA und Kanada

Der technische Support von Blink, Inc. US ist telefonisch unter (804) 784-2347 in der Zeit von 9.30h bis 17.00h (Eastern Standard Zeit) erreichbar, sowie per Fax unter (804) 784-2357.

Telefonische Unterstützung wird für die Zeit von 90 Tagen nach Registrierung kostenlos angeboten. Dies gilt für jede Anfrage, die im direkten Bezug zu Blinker steht. Nach dieser Zeit können Sie Mitglied im Blinkinc Developers’ Club zu einem Nominalbeitrag werden, der Ihnen freien Fax-Support sowie zwischenzeitliche Updates auf dem Postweg bietet, sobald diese zur Verfügung stehen. Rufen Sie uns für weitere Einzelheiten an.

Registrierung und Technische Unterstützung

Europäische Kunden

Wenn Ihrer Seriennummer ein ‘ER’ vorangestellt ist, dann haben Sie 90 Tage lang Anspruch auf kostenlosen technischen Support durch unsere europäischen Fachhändler.

Beginnt Ihre Seriennummer mit ‘BR’, so sind Sie im Besitz eines "Grauimports" bzw. einer amerikanischen Version von Blinker. Sie haben dann keinen Anspruch auf kostenlose technische Unterstützung oder Upgrades in Europa, bis sie die Software auf eine europäische Version upgraden. Bitte wenden Sie sich an Blinkinc für weitere Details.

Wir haben unser Möglichstes getan, um die Preisdifferenz zwischen den Kosten unserer Software-Lizenz in den USA und in Europa zu verringern, können aber die ‘Grau-Importe’ aus den USA nicht verhindern. Wir raten Ihnen dringend, sich deswegen an den betroffenen Distributor zu wenden, falls Sie dadurch Schwierigkeiten haben.

Registrierung und Technische Unterstützung

Übersee

Benutzer außerhalb der USA oder Kanada werden gebeten, den örtlichen oder zuständigen Fachhändler anzusprechen, der Ihnen als registriertem Kunden 90 Tage lang technische Unterstützung zu Blinker geben wird. Dies ist mit sämtlichen Vertragshändlern in Übersee so vereinbart.

Registrierung und Technische Unterstützung

Anschrift in den USA

Unsere Anschrift in den USA lautet:

Blinkinc
P.O. Box 29858
Richmond, Virginia 23242-0858
U.S.A.

Lizenzabkommen

WICHTIG - LESEN SIE DIE LIZENZVEREINBARUNG BITTE SORGFÄLTIG DURCH; BEVOR SIE DAS VERSIEGELTE DISKETTENPAKET ÖFFNEN. Mit dem Öffnen des versiegelten Diskettenpaketes erkennen Sie die Bedingungen dieser Blink, Inc. Endverbraucher-Lizenz an.

BLINK, INC. ENDVERBRAUCHER-LIZENZABKOMMEN

Dies ist eine rechtliche Vereinbarung (nachfolgend ‘Vereinbarung’ genannt) zwischen Ihnen, dem Endverbraucher, und Blink, Inc. (nachfolgend ‘BLINK’ genannt). MIT DEM ÖFFNEN DES VERSIEGELTEN DISKETTENPAKETES ERKENNEN SIE DIE BEDINGUNGEN DIESER VEREINBARUNG AN. WENN SIE DEN LIZENZVEREINBARUNGEN NICHT ZUSTIMMEN; SENDEN SIE BITTE DAS UNGEÖFFNETE DISKETTENPAKET UND DAS BEGLEITMATERIAL (inclusive aller schriftlichen Unterlagen und Verpackungen) INNERHALB VON 30 TAGEN NACH ERHALT DES PRODUKTES GEGEN ERSTATTUNG DES KAUFPREISES AN IHREN HÄNDLER ZURÜCK:

1. Lizenz BLINK gewährt dem Anwender ein einfaches, nicht übertragbares Recht zur Benutzung der urheberrechtsgeschützten Software ein. Dieses Recht gilt für den Gebrauch einer (1) Kopie dieses Paketes (nachstehend ‘Software’ genannt), enthalten auf dem Diskettenpaket und auf einem einzigen Computer. Dies gilt ebenso für die begleitenden Unterlagen (nachstehend ‘Dokumentation’ genannt) der Software in Übereinstimmung mit der Lizenzvereinbarung. Die Software darf nicht in einem Netzwerk und / oder auf mehreren Computern installiert werden und so mehrerern Teilnehmern die Benutzung ermöglichen. Diese Lizenz gilt nur für einen einzigen Benutzer. Jeder weitere Programmierer muß eine separate Lizenz dieser Software bei BLINK erwerben. Ferner gewährt Ihnen BLINK das Recht, mit Hilfe der Software ausführbare Programme zu erstellen und diese zu vertreiben. Es muß jedoch dafür gesorgt werden, daß nicht Teile der Software aus den ausführbaren Programmen extrahiert werden können.

2. Copyright und andere Eigentumsrechte Die Software und Dokumentation sind Eigentum von BLINK, geschützt durch die Urheberrechte der USA und internationale Vertragsvereinbarungen, sowie möglicherweise durch andere Gesetze. Sie dürfen die Software (a) für Sicherungs- und Archivzwecke einmalig kopieren oder (b) auf eine Festplatte oder ein anderes Speichermedium einer Computers übertragen. Jede Kopie muß die Copyright-Meldung von BLINK und andere Hinweise auf Urheberrechte beinhalten. Sie dürfen die Dokumentation nicht kopieren. Alle Rechte an der Software, Kopien der Software und der Dokumentation verbleiben bei BLINK. BLINK behält sich das Recht vor, die Software oder die Dokumentation zu ändern und / oder zu verbessern, ohne dabei die Verpflichtung einzugehen, Sie von diesen Änderungen in Kenntnis zu setzen oder solche Änderungen Ihnen unaufgefordert zukommen zu lassen.

3. Zusätzliche Einschränkungen Software und Dokumentation dürfen nicht verliehen, vermietet, geleast oder auf andere Weise weitergegeben werden. Weder die Software noch Kopien davon noch die Dokumentation sind ohne schriftliche Genehmigung von BLINK übertragbar. Anfragen bzgl. solcher Übertragung sollten an BLINK unter Verwendung der in dieser Vereinbarung angegebenen Adresse gerichtet werden. Dieser Antrag auf Lizenzübertragung sollte enthalten: (a) Name und Anschrift der Person, die als neuer Lizenznehmer vorgeschlagen wird und an die Software und Dokumentation übergehen sollen; (b) eine Bestätigung, daß Sie, der jetzige Lizenznehmer, weder Kopien von der Software noch der Dokumentation nach dem Lizenzübergang behalten werden; und (c) eine unterschriebene Bestätigung des vorgeschlagenen neuen Lizenznehmers, daß er diese Lizenzvereinbarung ohne Einschränkung anerkennt. Jeder nicht von BLINK autorisierte Versuch, Sublizenzen zu vergeben, oder Rechte, Auflagen und Verpflichtungen dieser Vereinbarung an Dritte zu übertragen ist ungültig und unwirksam. Reverse-Engineering, Dekompilierung oder Disassemblierung der Software sind nicht

erlaubt. Auch die Änderung, Übersetzung oder Erzeugung abgewandelter Formen der Software und Dokumentationen sind nicht erlaubt. Sie stimmen zu, Software und Dokumentation vor ungenehmigter Veröffentlichung, Verwendung, Reproduktion und / oder der Weitergabe zu schützen.

ES IST NICHT ERLAUBT, DIE SOFTWARE, KOPIEN DER SOFTWARE UND / ODER DER DOKUMENTATION GANZ ODER TEILWEISE IN ANDERER FORM ZU VERWENDEN, ZU KOPIEREN ODER ZU ÜBERTRAGEN ALS ES AUSDRÜCKLICH IN DIESER VEREINBARUNG GESTATTET IST.

4. Dual Media Wenn die Diskettenverpackung sowohl 3,5- als auch 5,25-Zoll Disketten enthält, dürfen Sie nur die für Ihren Einzelplatz-Computer geeigneten Disketten benutzen. Die andere Diskette darf weder auf einem anderen Computer verwendet werden, noch darf sie verliehen, vermietet, verleast, vertrieben oder an Dritte übertragen werden.

5. Dauer Diese Lizenz ist ab dem Datum gültig, an dem Sie Software und Dokumentation erhalten haben, und endet, wenn Sie den kompletten Inhalt des Paketes, sowie alle davon erstellten Kopien vernichtet haben. Diese Lizenz endet ferner in dem Moment, in dem der Lizenznehmer einen Punkt oder eine Bedingung dieser Lizenzvereinbarung verletzt. (a) BLINK behält sich in diesem Fall eine Geltendmachung von Schadenersatzansprüchen vor. (b) Außerdem muß der Lizenznehmer in diesem Fall den kompletten Inhalt dieses Paketes sowie alle davon erstellten Kopien an BLINK zurückgeben oder vernichten. Der Lizenznehmer kann in diesem Fall gegenüber der Firma BLINK keine Ansprüche gleich welcher Art geltend machen.

6. Beschränkte Haftung BLINK garantiert dem Original-Lizenznehmer, daß die gelieferten Disketten selbst, auf denen sich die Software befindet, (nicht die darauf befindliche Software) keine Material- und / oder Herstellungsmängel aufweisen und dreißig (30) Tage nach Erhalt der Ware bei normaler Verwendung genutzt werden können.

ÜBER DIE OBEN BESCHRIEBENE GARANTIE HINAUS WIRD VON BLINK KEINE WEITERE GEWÄHRLEISTUNG JEDWEDER ART GEGEBEN. DES WEITEREN SCHLIESST BLINK ALLE WEITERE GEWÄHRLEISTUNG AUS WIE Z.B. VERKAUFSFÄHIGKEIT ODER EIGNUNG FÜR BESTIMMTE ZWECKE DES ERWERBERS. AUCH WIRD GEWÄHRLEISTUNG, DIE AUS LAUFZEITVERHALTEN, HANDEL ODER HANDELSGEBAREN ABGELEITET WIRD, AUSGESCHLOSSEN. DIE AUF DER / DEN DISKETTE(N) BEFINDLICHE SOFTWARE UND DIE DOKUMENTATION WERDEN OHNE WEITERE ZUSÄTZE UND OHNE JEGLICHE GEWÄHRLEISTUNG GELIEFERT.

BLINK GEWÄHRLEISTET NICHT, DASS DIE SOFTWARE ODER DIE DOKUMENTATION IHREN ANFORDERUNGEN GENÜGT, DASS DIE SOFTWARE OHNE UNTERBRECHUNG UND OHNE FEHLER ARBEITET, ODER DASS ALLE MÄNGEL IN DER SOFTWARE UND / ODER DER DOKUMENTATION BESEITIGT WERDEN. SIE ÜBERNEHMEN DIE ALLEINIGE VERANTWORTUNG FÜR DIE AUSWAHL DER SOFTWARE UND DER DOKUMENTATION. DAS GESAMTE RISIKO BZGL. QUALITÄT, LAUFZEITVERHALTEN UND DER ERGEBNISSE, DIE MIT SOFTWARE UND DOKUMENTATION ERREICHT WERDEN, IST AUF IHRER SEITE. SOLLTE SICH DIE SOFTWARE ODER DOKUMENTATION ALS FEHLERHAFT ERWEISEN, HABEN SIE UND NICHT BLINK ODER DEREN HÄNDLER DIE KOSTEN ALLER NOTWENDIGEN DIENSTLEISTUNGEN, REPARATUREN ODER VERBESSERUNGEN ZU TRAGEN.

KEINE MÜNDLICH ODER SCHRIFTLICH GEGEBENE INFORMATION ODER RATSCHLÄGE, DIE VON BLINK, SEINEN HÄNDLERN ODER DISTRIBUTOREN GEGEBEN WURDEN, WIRD NEUE GEWÄHRLEISTUNGSANSPRÜCHE SCHAFFEN ODER DEN GELTUNGSRAHMEN DIESER BESCHRÄNKTEN GEWÄHRLEISTUNG ERWEITERN. SOLCHE INFORMATIONEN DÜRFEN NICHT ALS GEWÄHRLEISTUNG VERSTANDEN WERDEN.

IST DER AUSSCHLUSS WEITERGEHENDER GEWÄHRLEISTUNG DURCH DAS ANWENDBARE RECHT NICHT GESTATTET, WERDEN ALLE GESETZLICH AUFERLEGTEN GEWÄHRLEISTUNGEN, EINSCHLIESSLICH VERKAUFBARKEIT, EIGNUNG FÜR EINEN BESTIMMTEN ZWECK HIERMIT AUF DIE DAUER VON DREISSIG (30) TAGEN NACH ERHALT DER WARE BESCHRÄNKT.

EINIGE STAATEN ERLAUBEN KEINE ZEITLICHE BESCHRÄNKUNG AUFERLEGTER GEWÄHRLEISTUNG. IN SOLCHEN FÄLLEN KOMMEN DIE OBEN GENANNTEN EINSCHRÄNKUNGEN BEI IHNEN NICHT ZUR ANWENDUNG. DIESE BESCHRÄNKTE GEWÄHRLEISTUNG GIBT IHNEN BESONDERE JURISTISCHE RECHTE. WEITERE RECHTE KÖNNEN VON STAAT ZU STAAT VERSCHIEDEN SEIN.

7. Beschränkte Haftung für Folgeschäden BLINK's gesamte Haftungsverpflichtung und Ihr ausschließlicher Anspruch unter dieser Vereinbarung beschränkt sich auf zwei Möglichkeiten, von denen BLINK sich vorbehält, welche zur Anwendung kommt. Diese Möglichkeiten sind: (a) Ersatz der Diskette, die die 'beschränkte Gewährleistung' von BLINK nicht erfüllt hat und an den Händler mit einer Rechnungskopie zurückgegangen ist, von dem Sie die Ware erhalten haben; oder (b) Erstattung des Preises, den Sie für die Lizenz gezahlt haben, Software und Dokumentation zu nutzen, wenn BLINK nicht in der Lage ist, eine Ersatzdiskette frei von Material- und Herstellungsfehlern zu liefern.

IN KEINEM FALL SIND WEDER BLINK NOCH SEINE LIEFERANTEN FÜR IRGENDWELCHE SCHÄDEN ERSATZPFLICHTIG. UNEINGESCHRÄNKT DARIN EINGESCHLOSSEN SIND SCHÄDEN AUS ENTGANGENEM GEWINN, BETRIEBSUNTERBRECHUNG, VERLUST VON GESCHÄFTLICHEN INFORMATIONEN ODER VON DATEN, FINANZIELLEM VERLUST ODER ANDERE INDIREKTE, BESONDERE FOLGEN ODER MÖGLICHE SCHÄDEN, DIE AUS DEM GEBRAUCH ODER DER UNMÖGLICHKEIT DES GEBRAUCHS DER SOFTWARE UND / ODER DER DOKUMENTATION ENTSTEHEN, WIE AUCH IMMER DIESE SCHÄDEN ENTSTANDEN SIND UND UNTER BERÜCKSICHTIGUNG JEDER INTERPRETATION VON HAFTUNGSVERPFLICHTUNG, SELBST WENN BLINK VON DER MÖGLICHKEIT EINES SOLCHEN SCHADENS UNTERRICHTET WORDEN IST. AUF JEDEN FALL IST DIE HAFTUNG, WIE AUCH IMMER BEGRÜNDET, BESCHRÄNKT AUF DEN BETRAG, DEN SIE TATSÄCHLICH FÜR DIE LIZENZ BEZAHLT HABEN.

EINIGE STAATEN ERLAUBEN KEINEN HAFTUNGSAUSSCHLUSS ODER HAFTUNGSBEGRENZUNG FÜR FOLGESCHÄDEN. IN SOLCHEN FÄLLEN GELTEN DIE OBEN GENANNTEN BESCHRÄNKUNGEN UND AUSSCHLÜSSE NICHT FÜR SIE.

8. Exportbeschränkungen Sie bestätigen, daß Sie weder die Absicht haben, Kopien der Software und / oder der Dokumentation oder technische Daten, die in der Software und / oder Dokumentation enthalten sind, oder ein direktes Produkt davon, (direkt oder indirekt) in irgendein Land zu exportieren, in das die US-Regierung den Export verbietet, oder die Software und / oder Dokumentation für unerlaubte Zwecke einzusetzen oder den Einsatz zu nicht erlaubten Zwecken zu gestatten, noch dies tun werden.

9. RECHTLICHE AUFLAGEN DER U.S.-REGIERUNG Die Software und die Dokumentation werden unter Vorbehalt rechtlicher Beschränkungen überlassen. Beschränkungen in Anwendung, Duplizierung oder Bekanntmachung durch die US-Regierung sind geregelt in Subparagraph (c)(1)(ii) in der Bestimmung DFARS 252.227-7013 in 'The Rights in Technical Data and Computer Software' bzw. in den Subparagraphen (c)(1) und (2) der Bestimmung 48 CFR 52.227-19 in 'Commercial Computer Software - Resticted Rights', soweit anwendbar. Außerdem gelten Beschränkungen wie in dieser Vereinbarung auseinandergesetzt. Vertragspartner und Hersteller ist Blink, Inc., P.O. Box 29858, Richmond, Virginia 23242-0858, (804) 784-2087.

10. Allgemeines Diese Vereinbarung wird durch das Gesetz des Staates Virginia geregelt. Falls irgendein Absatz dieser Vereinbarung unwirksam ist oder nicht geltend gemacht werden kann, wird die Geltung der Vereinbarung im Übrigen nicht berührt und bleibt gemäß ihrer Zwecksetzung anwendbar. Wenn Sie

Fragen bzgl. der Vereinbarung haben oder aus anderen Gründen BLINK erreichen wollen, schreiben Sie an Blink, Inc., P.O. Box 29858, Richmond, Virginia 23242-0858, USA.

SIE BESTÄTIGEN, DASS SIE DIESE VEREINBARUNG GELESEN UND VERSTANDEN HABEN UND IHR OHNE EINSCHRÄNKUNG ZUSTIMMEN. AUSSERDEM BESTÄTIGEN SIE, DASS SIE DIE EINZIGE UND VOLLSTÄNDIGE VEREINBARUNG ZWISCHEN IHNEN (LIZENZNEHMER) UND BLINK IST UND ALLE BISHERIGEN DEN GEGENSTAND DER VEREINBARUNG BETREFFENDEN MÜNDLICHEN UND SCHRIFTLICHEN VEREINBARUNGEN UND VERTRÄGE DAMIT UNGÜLTIG WERDEN. EINE ÄNDERUNG DIESER VEREINBARUNG BEDARF DER SCHRIFTFORM UND DER UNTERSCHRIFT VON IHNEN UND BLINK.

2

Schnelleinstieg

Überblick

Umgebungsvariablen

Blinker ausführen

Link-Skript-Dateien

Die Blinker-Hilfsprogramme

Make-Programme

DOS-Extender-Programm

Komprimierte .EXEs und .DLLs

Schnelleinstieg für verschiedene Compiler

Überblick

Wie Sie in diesem Kapitel vorgehen:

- Lesen Sie die ersten Abschnitte zur Umgebungskonfiguration und zum Aufruf von Blinker.
- Blättern Sie zu dem für Ihren Compiler relevanten Abschnitt in diesem Kapitel.
- Nutzen Sie die aufgeführten Beispiele, um schnellstmöglich mit Blinker vertraut zu werden.
- Lesen Sie das Kapitel ‘Blinker-Befehle’ und binden Sie zusätzliche Blinker-Eigenschaften in Ihr Programm mit ein.

Schnelleinstieg für verschiedene Compiler

Überblick

Unterstützte Compiler

Alaska Xbase++
Borland Turbo Assembler
Borland Turbo C / Turbo C++
Borland C++
CA-Clipper Sommer '87
CA-Clipper 5.x
FORCE
Microsoft Assembler
Microsoft Visual Basic for DOS
Microsoft BASIC PDS 7.1
Microsoft Quick BASIC 4.5
Microsoft C / C++ / Quick C
Microsoft FORTRAN
Microsoft PASCAL
Microsoft Visual C++
Stony Brook Modula 2
Stony Brook Pascal+
Symantec C++
WATCOM C++
Zinc Application Framework für DOS

HINWEIS: Obwohl die Dokumentation nur die speziellen Versionen der o.g. Compiler beschreibt, ist es möglich, daß frühere Versionen dieser Compiler und auch solche, die das Intel/Microsoft .OBJ-Datei-Format erzeugen, unterstützt werden.

Umgebungsvariablen

Umgebungsbereich

Die Umgebungsvariable PATH

Die Umgebungsvariable OBJ

Die Umgebungsvariable LIB

Die Umgebungsvariable TMP oder TEMP

Die Umgebungsvariable BLINKER

Umgebungsvariablen

Umgebungsbereich

DOS-Umgebungsvariablen werden benutzt, um Parameter an Programme zu übergeben, die gestartet werden. Dies bedeutet, daß diese Informationen nicht mehr in der Kommandozeile oder über Parameterdateien übergeben werden müssen.

Der Platz für den Umgebungsbereich wird von DOS beim Starten des Computers reserviert. Sollte also eine Meldung erscheinen, die aussagt, daß nicht ausreichend Platz im Umgebungsbereich vorhanden ist, so fügen Sie bitte folgende Zeile in Ihre CONFIG.SYS-Datei ein und führen Sie anschließend einen Neustart durch:

```
SHELL=C:\COMMAND.COM /P /E:n
```

Die SHELL-Anweisung teilt DOS mit, welches Programm als Kommandozeilen-Interpreter verwendet werden soll, um den DOS-Prompt anzuzeigen und um Programme wie ‘COPY’ usw. zu starten.

Der Parameter /P besagt, daß die COMMAND.COM resident im Speicher verbleiben soll, d.h., daß es sich nicht um eine sekundäre Kopie handelt, die von einem anderen Programm geladen wurde.

Der Parameter /E:n gibt an, wieviel Platz für den Umgebungsbereich von COMMAND.COM reserviert werden soll. Die Angabe ‘n’ erfolgt dabei in Bytes. Sollten Sie DOS 3.1 verwenden, so wird durch diese Anweisung die Anzahl von 16-Bytes-Blöcken festgelegt. Der Wert sollte groß genug gewählt werden, um die folgenden Umgebungsvariablen aufzunehmen. Wollen Sie eine Umgebungsvariable erzeugen, dann geben Sie SET, gefolgt von dem Variablennamen, ein. Direkt nach dem Variablennamen muß dann ein ‘=’ und der zuzuweisende Wert folgen.

DOS selbst benutzt zum Beispiel die PATH-Umgebungsvariable als Liste von Verzeichnissen, in denen DOS nach einer Datei suchen soll, wenn ein Programm von der Kommandozeile zu starten ist. Um DOS mitzuteilen, daß die Suche zunächst im C:\ Verzeichnis und anschließend im Verzeichnis C:\BLINKER durchgeführt werden soll, müßten Sie am DOS-Prompt folgende Anweisung eingeben:

```
SET PATH=C:\;C:\BLINKER;
```

Sie können Umgebungsvariablen auch über Batch-Dateien wie der Autoexec.bat setzen, indem Sie die gleiche Kommandozeile, die Sie auch am DOS-Prompt eingeben würden, in diese Datei einfügen.

Umgebungsvariablen

Die Umgebungsvariable PATH

Idealerweise sollte die PATH-Variable so gesetzt sein, daß diese das BIN Verzeichnis aus dem Verzeichnis, in dem Blinker installiert wurde, mit einschließt. Die Programmdateien BLINKER.COM, BLINKER.DLL, BLINKER.EXE und BLILIB.EXE müsssen alle in einem über PATH spezifizierten Verzeichnis zu finden sein.

Umgebungsvariablen

Die Umgebungsvariable OBJ

Die Umgebungsvariable OBJ kann genauso gesetzt werden wie die PATH-Variable. Diese teilt Blinker mit, wo Objektdateien des FILE Befehls gesucht werden sollen, falls sie sich nicht im aktuellen Verzeichnis befinden. Blinker durchsucht auch die Verzeichnisse der LIB Umgebungsvariable, falls .OBJ Dateien nicht im aktuellen Verzeichnis oder in den Verzeichnissen von OBJ gefunden werden.

Sie kann benutzt werden, um alternative Pfade anzuzeigen, in denen Objektdateien zu finden sind, die sonst nirgendwo im Zugriff sein sollen. Auch kann der Pfad auf eine RAM-Disk gelegt werden, was die Linkgeschwindigkeit steigert. Desweiteren kann so das wiederholte Eingeben des Objektpfades in einer FILE - Anweisung entfallen.

Umgebungsvariablen

Die Umgebungsvariable LIB

Die Umgebungsvariable LIB kann wiederum eine Liste von Zugriffspfaden enthalten. Blinker wird so mitgeteilt, in welchen Verzeichnissen nach den .LIB-Dateien gesucht werden soll, die durch den LIBRARY- oder SEARCH-Befehl angesprochen werden, wenn sie sich nicht im aktuellen Verzeichnis befinden. Blinker durchsucht die Verzeichnisse der LIB Umgebungsvariable auch nach .OBJ Dateien, die nicht im aktuellen Verzeichnis oder in den Verzeichnissen von OBJ gefunden werden können.

Auch hier kann ebenfalls ein Verweis auf eine RAM-Disk enthalten sein, um die Linkgeschwindigkeit zu steigern. Außerdem kann man sich die mehrfache Angabe eines Pfades innerhalb des LIB-Befehls ersparen.

Umgebungsvariablen

Die Umgebungsvariable TMP oder TEMP

Die Umgebungsvariable TMP kann dazu eingesetzt werden, Blinker das Verzeichnis vorzugeben, in dem er zur Linkzeit temporäre Dateien anlegen kann. Diese Einstellung überschreibt den Blinker-Link-Skriptbefehl ‘WORKFILE’. Alternativ kann auch die Umgebungsvariable TEMP verwendet werden, wenngleich ‘TMP’ vorrangig behandelt wird.

Werden sehr große Programme gelinkt, kann es vorkommen, daß der von Blinker benötige Speicherbedarf den verfügbaren Speicher überschreitet. In diesem Fall versucht Blinker, bestimmte Daten temporär auf die Platte auszulagern. Für optimale Leistungswerte sollte die Umgebungsvariable TMP auf eine RAM-Disk verweisen.

Auch der DOS-Extender von Blinker verwendet diese Variablen, um festzulegen, wo seine virtuelle Speicherauslagerungsdatei abgelegt werden kann, wenn zur Laufzeit zuwenig Hauptspeicher zur Verfügung steht.

Umgebungsvariablen

Die Umgebungsvariable BLINKER

Beim Start eines Programms überprüft der Blinker-Kernel die DOS-Umgebung auf die Existenz der Umgebungsvariable BLINKER, mit der bestimmte Blinker Einstellungen zur Laufzeit überschrieben werden können. Der Name BLINKER kann durch das Kommando BLINKER ENVIRONMENT NAME geändert werden. Weitere Einzelheiten dazu finden Sie im Abschnitt "BLINKER ENVIRONMENT NAME" der Online-Hilfe im Kapitel 9. Ist der Name der Umgebungsvariable vorhanden, dann wird die Variable auf folgende Einstellungen hin untersucht:

/CEnnn,nn	Größe des EMS für Overlay-Pufferung
/CXnnn,nn	Größe des XMS für Overlay-Pufferung
/DHn	Debug-Handle
/DI3	Debug-Interrupt 3
/DTnnn	Grenzwert für Dual Mode in KB
/EBnnn	Blockgröße des Erweiterungsspeichers (KB)
/ED-	Extender-Debugging ausschalten
/ED+	Extender-Debugging einschalten
/EInnn,nnn	Anzahl der Extender IPX Puffer.
/EK-	Spezielle Tastastaturbehandlung im Extender ausschalten
/EK+	Spezielle Tastastaturbehandlung im Extender einschalten
/ELnnn	Erweiterungsspeicherbegrenzung in KB
/ETnnn	Extended Mode Schwellwert in KB
/HD-	Deaktiviert Unterstützung für DPMI-Host
/HD+	Aktiviert Unterstützung für DPMI-Host

/HM-	Statusmeldung beim Programmstart ausschalten
/HM+	Statusmeldung beim Programmstart einschalten
/HP-	Übergehen von IRQ-Interrupts im Host ausschalten
/HP+	Übergehen von IRQ-Interrupts im Host einschalten
/HQ-	Deaktiviert Unterstützung für QDPMI-Host
/HQ+	Aktiviert Unterstützung für QDPMI-Host
/HV-	Deaktiviert Unterstützung für VCPI-Host
/HV+	Aktiviert Unterstützung für VCPI-Host
/HX-	Deaktiviert Unterstützung für XMS-Host
/HX+	Aktiviert Unterstützung für XMS-Host
/LRnnn	Speicher < 1 MB für Programm reservieren
/OOnnn	Arbeitsgröße des Overlay-Bereichs in KB
/OP-	Benutzung von EMS für Overlays verbieten
/OP+	Benutzung von EMS für Overlays erlauben
/OU-	Benutzung von XMS-UMBs für Overlays verbieten
/OU+	Benutzung von XMS-UMBs für Overlays erlauben
/SD<pfad>	Verzeichnis für Swapdatei der Funktion SWPRUNCMD()
/SE-	Benutzung von EMS durch SWPRUNCMD() verbieten
/SE+	Benutzung von EMS durch SWPRUNCMD() erlauben
/SU-	UMB-Benutzung durch SWPRUNCMD() verbieten
/SU+	UMB-Benutzung durch SWPRUNCMD() erlauben
/SX-	Benutzung von XMS durch SWPRUNCMD() verbieten
/SX+	Benutzung von XMS durch SWPRUNCMD() erlauben
/WS-	Erkennung von SoftICE/W ausschalten
/WS+	Erkennung von SoftICE/W einschalten

Die Anweisung /CEnnn,nn überschreibt die Linkanweisung BLINKER CACHE EMS nnn, welche die Menge an EMS-Speicher angibt, die der Overlay-Manager für die Overlay-Pufferung verwenden darf. Eine genauere Beschreibung zur Benutzung dieser Werte finden Sie in Kapitel 9 unter "BLINKER CACHE EMS".

Die Anweisung /CXnnn,nn überschreibt die Linkanweisung BLINKER CACHE XMS nnn. Diese gibt die Menge an XMS-Speicher an, die der Overlay-Manager für die Overlay-Pufferung nutzen darf. Eine genauere Beschreibung zur Benutzung dieser Werte finden Sie in Kapitel 9 unter "BLINKER CACHE XMS".

Die /DHn Anweisung überschreibt die Linkanweisung BLINKER DEBUG HANDLE n, welche den Handle der Zielausgabe der Debug-Informationen festlegt. Eine genauere Beschreibung zur Benutzung dieser Werte finden Sie in Kapitel 9 unter "BLINKER DEBUG HANDLE".

Die Einstellung von /DI3 ermöglicht die Erzeugung einer Interrupt 3 Debug-Anweisung, unmittelbar bevor der Blinker-Kernel auf die erste Zeile des Hauptprogrammes springt. Dieser Schalter sollte nur während eines Debug-Vorganges gesetzt werden. Mehr Einzelheiten hierzu in der Online-Hilfe im Abschnitt über Debuggen.

/DTnnn: Mit dieser Einstellung kann man zur Laufzeit einen Grenzwert für den Dual Mode, der zur Linkzeit festgelegt wurde, überschreiben. Diese Möglichkeit kann man auch in der Testphase nutzen, wenn man z.B. erzwingen will, daß ein Programm im Real Mode läuft. In diesem Fall setzen Sie z.B. /DT20000. Unter der Beschreibung zu BLINKER EXECUTABLE DUAL in der Onlin-Hilfe finden siehe weitere Einzelheiten.

Die Anweisung /EBnnn überschreibt den Linkbefehl BLINKER EXTMEM BLOCK, der zur Steuerung der Größe der vom Blinker-Extender verwendeten Speicherbelegungsblöcke genutzt wird. Weitere

Einzelheiten finden Sie unter dem Eintrag zum Befehl BLINKER EXTMEM BLOCK in der Online-Hilfe.

Die Einstellung von /EDc (wobei c=- für ausgeschaltet und c=+ für eingeschaltet steht) steuert den Debug-Modus des Blinker-Extenders. Diese Einstellungen schalten bestimmte Debugger-Meldungen ein bzw. aus, die vom DOS-Extender geliefert werden, wie z.B. die Überprüfung auf Bezeichner von Segmenten, die nicht gesperrt oder vorhanden sind, oder die Freigabe von Selektoren durch den Aufruf der Funktion ‘Speicher freigeben’. Es empfiehlt sich, Ihre Programme mit dieser eingeschalteten Option unter einem VCPI- oder einem XMS-Host zu testen (d.h. kein DPMI), um diese Fälle zu überprüfen. Die Voreinstellung ist ‘ausgeschaltet’.

Der Schalter /Einnn bestimmt die Anzahl der IPX ECB-Puffer, die durch das IPX-Modul des Extenders bereitgestellt werden. Der Vorgabewert ist 32. Einzelheiten hierzu finden unter dem Befehl BLINKER EXECUTABLE IPX in der Online-Hilfe.

Die Einstellung /EKc (wobei c=- für ausgeschaltet und c=+ für eingeschaltet steht) steuert die Tastaturbehandlung des Blinker-Extenders. Diese Einstellungen schalten eine spezielle Tastaturbehandlungsroutine für bestimmte BIOS-Aufrufe der Tastatur durch den Extender ein bzw. aus, um unnötige Modus-Umschaltungen zu vermeiden. Wenn eingeschaltet, kann diese Option die Leistung von seriellen Kommunikationsroutinen erhöhen. Wenn ausgeschaltet, bleibt die Kompatibilität zu älteren speicherresidenten Programmen erhalten, die die Tastatur behandeln. Vorgabewert ist ausgeschaltet.

Die /ELnnn Anweisung überschreibt den Linkbefehl BLINKER EXTMEM LIMIT, über den die Gesamtgröße des vom Blinker-Extender belegten Erweiterungsspeichers gesteuert wird. Einzelheiten finden Sie unter dem Eintrag zu dem Befehl BLINKER EXTMEM LIMIT in der Online-Hilfe.

/ETnnn: Mit dieser Einstellung zur Laufzeit kann die Extended-Mode-Grenze, die zur Linkzeit angegeben wurde, überschrieben werden. Weitere Einzelheitenfinden Sie unter dem Befehl BLINKER EXECUTABLE EXTENDED in der Online-Hilfe.

Die /HDc Anweisung (c=- deaktiviert and c=+ aktiviert) überschreibt den Linkbefehl BLINKER HOST DPMI, durch den die Verwendung eines DPMI-Host durch den Blinker-Extender gesteuert wird. Weitere Einzelheiten finden Sie unter dem Eintrag zu dem Befehl BLINKER HOST DPMI in der Online-Hilfe.

/HMc (c=- deaktiviert and c=+ aktiviert) : Mit diesen Einstellungen zur Laufzeit kann man die Anzeige einer Statusmeldung beim Programmstart ein- bzw. ausschalten. Unter der Beschreibung zum Befehl BLINKER HOST QDPMI finden siehe weitere Einzelheiten in der Online-Hilfe.

Die Einstellung /HPc (wobei c=- ausgeschaltet und c=+ eingeschaltet bedeutet) überschreibt den zur Linkzeit gegebenen Befehl BLINKER HOST PASSUP, der die automatische Übergabe von IRQ-Interrupts aus dem Real Mode an den Protected Mode durch den Blinker DOS-Extender steuert. Diese Einstellung sollte nur zu Debug-Zwecken nötig sein, z.B. wenn Interrupt-Behandlungroutinen getestet werden, und ist standardmäßig eingeschaltet. Mehr Einzelheiten hierzu unter dem Befehl BLINKER HOST PASSUP in der Online-Hilfe.

/HQc (c=- deaktiviert and c=+ aktiviert): Mit diesen Einstellungen zur Laufzeit kann man die Verwendung eines QDPMI-Host ein- bzw. ausschalten. Unter der Beschreibung zum Befehl BLINKER HOST QDPMI finden siehe weitere Einzelheiten in der Online-Hilfe.

Die /HVc Einstellung (c=- deaktiviert and c=+ aktiviert) überschreibt den Linkbefehl BLINKER HOST VCPI, durch den die Verwendung eines VCPI-Host durch den Blinker-Extender gesteuert wird. Weitere Einzelheiten finden Sie unter dem Eintrag zu dem Befehl BLINKER HOST VCPI in der Online-Hilfe.

Die /HXc Einstellung (c=- deaktiviert und c=+ aktiviert) überschreibt den Linkbefehl BLINKER HOST XMS, durch den die Verwendung eines XMSH-Host durch den Blinker-Extender gesteuert wird. Weitere Einzelheiten finden Sie unter dem Eintrag zu dem Befehl BLINKER HOST XMS in der Online-Hilfe.

Der Schalter /LRnnn überschreibt den zur Linkzeit festgelegten Blinker-Befehl BLINKER LOWMEM RESERVE nnn, der die Größe des konventionellen Speichers für die Verwendung eines Programmes bestimmt und verhindert, daß Blinker's DOS-Extender ihn für andere Zwecke benutzt. Mehr Informationen zur Verwendung dieses Wertes finden Sie in der Befehlsbeschreibung zu BLINKER LOWMEM RESERVE in der Online-Hilfe.

Die /OOnn Anweisung überschreibt die Linkanweisung BLINKER OVERLAY OPSIZE nn, welche die Arbeitsgröße der Blinker-Overlaybereiche festlegt. Für eine genauere Beschreibung dieser Werte sehen Sie bitte unter BLINKER OVERLAY OPSIZE in der Online-Hilfe nach.

Die /OPc Anweisung (wobei c=0 ein und c=+ aus bedeuten) überschreibt die Linkanweisung BLINKER OVERLAY PAGEFRAME, welche die Speicherbelegung für den Overlay-Bereich unter Verwendung von EMS (3.2+) kontrolliert. Vergleiche BLINKER OVERLAY PAGEFRAME in der Online-Hilfe.

Die /OUc Anweisung (wobei c=- ein und c=+ aus bedeuten) überschreibt die Linkanweisung BLINKER OVERLAY UMB, welche die Speicherbelegung für den Overlay-Bereich unter Verwendung von XMS 2.0 kontrolliert. Für eine genauere Beschreibung dieser Werte sehen Sie bitte in der Online-Hilfe unter BLINKER OVERLAY UMB nach.

Beachten Sie: Der Blinker Overlay-Manager übernimmt die Kontrolle, bevor das Programm startet. Dies bedeutet, daß die Systemressourcen, die durch diese Anweisungen kontrolliert werden, vor dem Start des Programms belegt werden.

Über die Umgebungsvariable BLINKER kann das Verhalten der in Blinker integrierten Swapfunktion (SWPRUNCMD()) beeinflußt werden. Dabei stehen Ihnen folgende Einstellungen zur Verfügung:

Die Anweisung /SD<pfad> kann zur Angabe eines Verzeichnisses verwendet werden, das von SWPRUNCMD() zum Speichern von Abbildern übergeordneter Programme auf die Festplatte benötigt wird. Diese Anweisung überschreibt den Parameter 'temporäres Verzeichnis', der an die Funktion SWPRUNCMD() übergeben wurde. Weitere Einzelheiten finden Sie unter dem Eintrag zu der Funktion SWPRUNCMD()in der Online-Hilfe.

Die /SEc Anweisung (c=- deaktiviert und c=+ aktiviert) überschreibt die Swapfunktion SWPUSEEMS(), die die Verwendung von EMS-Speicher zum Speichern von Speicherabbildern übergeordneter Programme durch SWPRUNCMD() steuert. Weitere Einzelheiten finden Sie unter dem Eintrag zu der Funktion SWPUSEEMS() in der Online-Hilfe.

Die /SUc Anweisung (c=- deaktiviertund c=+ aktiviert) überschreibt die Swapfunktion SWPUSEUMB(), die die Verwendung von UMB zur Auslagerung des residenten Teils des Funktionskerns der Swapfunktion durch SWPRUNCMD() steuert. Weitere Einzelheiten finden Sie unter dem Eintrag zu der Funktion SWPUSEUMB() in der Online-Hilfe.

Die /SXc Anweisung (c=- deaktiviert und c=+ aktiviert) überschreibt die Swapfunktion SWPUSEXMS(), die die Verwendung von XMS zum Speichern von Speicherabbildern übergeordneter Programme durch SWPRUNCMD() steuert. Weitere Einzelheiten finden Sie unter dem Eintrag zu der Funktion SWPUSEXMS() in der Online-Hilfe.

Der Schalter /WSc (wobei c=- ausgeschaltet und c=+ eingeschaltet bedeutet) aktiviert oder deaktiviert die Überprüfung auf den Soft-Ice/W-Debugger, der im Protected Mode in einer erweiterten DOS-Box unter Windows läuft. Voreinstellung ist eingeschaltet. Zu mehr Einzelheiten sehen Sie den Abschnitt 'NuMega Soft-Ice/W' in Kapitel 7 und die Datei DEBUG.WRI im Blinker-Verzeichnis.

Die Umgebungsvariable BLINKER kann eingesetzt werden, um Einstellungen schnell und einfach während des Testens eines Programmes zu ändern, oder auch, um unterschiedliche Einstellungen während der Entwicklung für individuelle Arbeitsplatzumgebungen zu ermöglichen.

Die einzelnen Schalter in der Umgebungsvariablen BLINKER müssen durch Leerzeichen getrennt sein,

z.B.:

```
SET BLINKER=/HM+ /ET2000
```

Dieses Beispiel ermöglicht die Anzeige von Host-Meldungen durch den Blinker-Extender und setzt den Schwellwert für Extended Memory auf 2000 KBytes.

Der Standardname ‘BLINKER’ der Umgebungsvariablen kann durch Verwendung des Befehls BLINKER ENVIRONMENT NAME geändert werden, wodurch Konflikte mit anderen Programmen, die ebenfalls mit Blinker gelinkt wurden und auf demselben Computer installiert sind, vermieden werden

So ändern Sie beispielsweise den Namen auf ‘MYAPP’, wodurch Blinker die Variable ‘BLINKER’ ignoriert und stattdessen ‘MYAPP’ verwendet:

```
SET MYAPP=/OP- /OO30 /CX128
```

Blinker ausführen

Linkbefehle und Spezifikationen für Dateien und Bibliotheken, die eingebunden werden sollen, können über die Kommandozeile angegeben werden. Gebräuchlicher ist es, sie in Link-Skriptdateien zu plazieren. Ebenso ist es möglich, beide Methoden miteinander zu kombinieren.

Blinker erkennt automatisch die Umgebung und kann sowohl als Windows- als auch als DOS-Programm laufen.

Von der DOS Kommandozeile aus

Blinker unter Windows ausführen

Blinker ausführen

Von der DOS Kommandozeile aus

BLINKER kann, ähnlich wie andere Linker, vom DOS-Prompt aus aufgerufen werden. Dabei werden in der Kommandozeile die erforderlichen Dateien und Linkbefehle genannt. Die gebräuchlichste Methode, BLINKER aufzurufen, ist:

```
BLINKER FILE Test
```

am DOS-Prompt einzugeben. FILE teilt dem BLINKER mit, daß die folgenden Angaben eine oder mehrere zu linkende Objektdateien enthalten. In diesem Fall ist TEST der Name der einzigen Datei (die Erweiterung .OBJ wird als Voreinstellung angenommen).

Dies kann um verschiedene andere Befehle erweitert werden:

```
BLINKER FILE Test LIB Mylib, Compilib NOBELL
```

Dabei teilt LIB dem BLINKER mit, daß die folgenden Angaben eine oder mehrere Bibliotheken sind, die mit eingebunden werden. In diesem Fall sind Mylib.LIB und Compilib.LIB (die Erweiterung .LIB wird als Voreinstellung angenommen) also Bibliotheken. NOBELL instruiert den BLINKER, daß kein Signalton auszugeben ist, wenn der Linkvorgang beendet ist.

Einige Blinker-Befehle beginnen mit dem Wort ‘BLINKER’, das bedeutet, daß das Wort ‘BLINKER’ wiederholt am DOS-Prompt erscheint:

```
BLINKER BLINKER INCREMENTAL OFF FILE Test
```

Das erste Wort ‘BLINKER’ startet die BLINKER.EXE. ‘BLINKER INCREMENTAL OFF’ ist ein einzelner Blinker-Befehl, der das inkrementale Linken ausschaltet. Über den Befehl ‘FILE’ wird die TEST.OBJ in das Programm eingebunden.

Beachten Sie, daß einige Blinker-Befehle nicht über die Kommandozeile angegeben werden können, sondern nur in einer Link-Skriptdatei (z.B. BEGINAREA / ENDAREA).

Blinker-Befehle, die spezielle Zeichen oder DOS-Begrenzer enthalten, sollten nicht über die Kommandozeile angegeben werden. Zum Beispiel müssen für die Angabe des Befehls BLINKER DEMONSTRATION DATE Schrägstriche (Slashes) verwendet werden. Schrägstriche aber werden vom

DOS-Kommandointerpreter ganz anders analysiert als ursprünglich vorgesehen. Daher können sich daraus Syntaxfehler oder ein unerwartetes Laufzeitverhalten ergeben.

Blinker ausführen

Blinker unter Windows ausführen

Um Blinker als Windows-Programm auszuführen, legen Sie für ihn einfach ein Symbol oder eine Verknüpfung in Verbindung mit der entsprechenden Befehlszeile an und Blinker wird als Windows-Programm ausgeführt, sobald Sie das Symbol anwählen. Sie können das Fenster ‘Taste drücken’, das in der Windows-Version nach Beenden des Linkvorgangs angezeigt wird, abschalten, indem Sie Ihrer Link-Skriptdatei den Befehl BATCH hinzufügen.

Unter Windows ’95/’98 und Windows NT werden Windows-Programme automatisch erkannt, daher läuft hier Blinker immer als Windows-Programm ab, selbst aus einer DOS-Box. Aus diesem Grund wird das Programm BLINKER.COM mitgeliefert, das die Ausführung von Blinker als DOS-Programm von der DOS-Befehlszeile aus ermöglicht. Geben Sie auf dem DOS-Prompt BLINKER ohne Erweiterung ein, führt DOS das Programm BLINKER.COM statt BLINKER.EXE aus, da es .COM- vor .EXE-Dateien sucht

Alternativ können Sie über das Programm BLICFG.EXE den Betriebssystemschalter im Programmheader von BLINKER.EXE auf Nur-DOS setzen, um eine Ausführung als Windows-Programm ganz zu unterbinden. Durch BLICFG /XD setzen Sie das Betriebssystem für Blinker auf Nur-DOS, über BLICFG /XW wird der Schalter wieder auf die Standardeinstellung Windows zurückgesetzt, was ein Betreiben von Blinker als Windows- und DOS-Programm wieder ermöglicht.

Wenn Sie Blinker als Windows-Programm ausführen, kann die gleiche Befehlszeilen-Syntax verwendet werden wie unter DOS, einschließlich der Umleitung von Blinker’s Ausgaben in eine Datei durch Angabe des Zeichens ‘>‘.

Link-Skript-Dateien

Verwendung einer Link-Skriptdatei

MS Link (Microsoft) Skriptdateien und Blinker

TLink (Borland) Skriptdateien und Blinker

WLink (WATCOM) Skriptdateien und Blinker

Skriptdatei und Kommandozeilenparameter

Verschachtelte Skriptdateien

Link-Skript-Dateien

Verwendung einer Link-Skript-Datei

Wenn Sie den Blinker einsetzen, werden Sie bald herausfinden, daß DOS, bedingt durch die Maximallänge von 128 Zeichen einer Kommandozeile, Grenzen setzt. Dies bedeutet, daß Sie nicht ausreichend Platz haben, um BLINKER die gewünschten Befehle zu geben. Diese Begrenzung kann man mit Hilfe einer Skriptdatei leicht umgehen. Eine Skriptdatei enhält alle Dateien und Befehle, die Blinker zur Verarbeitung benötigt. Sie macht wiederholte Eingaben am DOS-Prompt überflüssig. Die Skriptdatei kann mit einem beliebigen Texteditor oder einer Textverarbeitung erstellt werden. Jedoch muß der Texteditor in der Lage sein, die Datei im ASCII-Format abzuspeichern.

Die Link- oder Skriptdatei hat gewöhnlich den gleichen Namen wie das zu erstellende Programm. Sie trägt jedoch die voreingestellte Erweiterung .LNK. In diese Datei tragen Sie alle Befehle, Dateien und Bibliotheken ein, die Sie Blinker zum Linken vorgeben. Die Online-Hilfe enhält detaillierte Beschreibungen zu allen Befehlen, die entweder auf der Kommandozeilenebene oder in Skriptdateien eingesetzt werden können.

Um beispielsweise ein DOS Extended Programm für MS Visual C++ 1.5 zu erzeugen, könnte die Datei MYAPP.LNK folgende Eintragungen enthalten:

```
# Setzen des OUTPUT.EXE -Dateinamens
OUTPUT Myapp
# Setzen des Stacks auf 4 KB
STACK 4096
# eine DOS Extended .EXE erzeugen
BLINKER EXECUTABLE EXTENDED
# Objekt Dateien
FILE Myapp1
FILE Myapp2
FILE Myapp3
LIB Mylib
# erforderliche Ersatzbibliothek
SEARCH blxmvc15
# optionale Compiler-Bibliothek
LIB llibce
```

Es wäre offensichtlich Zeitverschwendung, diesen Wust an Informationen vor jedem Linklauf neu einzutippen. Aus diesem Grunde können Sie BLINKER mit Hilfe einer Linkdatei starten:

```
BLINKER @Myapp
```

Beachten Sie, daß die Benutzung des @-Symbols Blinker darüber informiert, daß jetzt der Name einer Linkdatei folgt. Dabei wird die Erweiterung .LNK erwartet.

Link-Skript-Dateien

MS Link (Microsoft) Skriptdateien und Blinker

Blinker erzeugt eine Datei BLINKER.INI, die sich im gleichen Verzeichnis wie BLINKER.EXE befindet. Diese INI-Datei enthält globale Einstellungen zur Steuerung des Verhaltens von Blinker zur Linkzeit. Der wichtigste Eintrag in dieser Datei ist die Zeile 'LinkScript=', die das Link-Skriptdateien-Format bezeichnet, das Blinker erwartet. Die Datei BLINKER.INI kann in das aktuelle Verzeichnis kopiert werden, wenn die globalen Einstellungen für ein spezielles Programm oder Projekt überschrieben werden sollen.

Die Voreinstellung von 'LinkScript' ist BLINKER, da die meisten Blinker-Eigenschaften nur über die leistungsstarken Freiformat-Link-Skriptdateien angewendet werden können.

Die Einstellung kann in MSLINK oder TLINK geändert werden (es wird nicht zwischen Groß- und Kleinbuchstaben unterschieden), wodurch alle Link-Skriptdateien im entsprechenden Format verarbeitet werden. Die meisten Optionen mit '/' werden von diesen Linkern auch unterstützt, nicht-unterstützte Optionen werden einfach übergangen.

Um beispielsweise eine MS LINK-Skriptdatei zu verwenden, muß die Datei BLINKER.INI folgenden Eintrag enthalten:

```
. . .
[General]
LinkScript=MSLINK
. . .
```

Diese Einstellung wirkt global und betrifft die Verarbeitung von allen Linkdateien, die an Blinker übergeben werden.

Um Blinker zu zwingen, eine einzelne Skriptdatei im MS Link Format zu bearbeiten, muß man den Parameter /QM zu Beginn der Kommandozeile von Blinker angeben. Um beispielsweise die Datei TEST.LNK als MS Link Skriptdatei zu bearbeiten, geben sie folgendes ein :

```
BLINKER /QM @TEST
```

Link-Skript-Dateien

TLink (Borland) Skriptdateien und Blinker

Der vorausgegangene Abschnitt ‘MS Link (Microsoft) Skriptdateien und Blinker’, beschreibt die Verwendung der ‘LinkScript’ Einstellung zur Steuerung der globalen Verarbeitung von Linkdateien.

Um beispielsweise eine TLINK-Skriptdatei zu verwenden, muß die Datei BLINKER.INI folgenden Eintrag enthalten:

```
. . .
[General]
LinkScript=TLINK
. . .
```

Diese Einstellung wirkt global und betrifft die Verarbeitung von allen Linkdateien, die an Blinker übergeben werden.

Um Blinker zu zwingen, eine einzelne Skriptdatei im TLink Format zu bearbeiten, muß man den Parameter /QT zu Beginn der Kommandozeile von Blinker angeben. Um beispielsweise die Datei TEST.LNK als TLink Skriptdatei zu bearbeiten, geben sie folgendes ein :

```
BLINKER /QT @TEST
```

Link-Skript-Dateien

WLink (WATCOM) Skriptdateien und Blinker

Um Blinker zu zwingen, eine einzelne Skriptdatei im WLink Format zu bearbeiten, muß man den Parameter /QW zu Beginn der Kommandozeile von Blinker angeben. Um beispielsweise die Datei TEST.LNK als WLink Skriptdatei zu bearbeiten, geben sie folgendes ein :

```
BLINKER /QW @TEST
```

Link-Skript-Dateien

Skriptdatei und Kommandozeilenparameter

Es ist möglich, daß Sie irgend wann die eingestellten Werte in der Skriptdatei übergehen möchten, ohne die Datei selbst zu ändern, oder sogar mit Hilfe einer Batchdatei Parameter an den Linker übergeben wollen. Dies geschieht ganz einfach, in dem Sie die Blinker-Befehle in der Kommandozeile und innerhalb der Skriptdatei(en) einsetzen. Das folgende Beispiel zeigt es.

Beispiel 1

Angenommen, ein Entwickler testet zahlreiche Programme, alle werden auf ähnliche Art kompiliert und

gelinkt. Das könnte durch Erstellen einzelner Skriptdateien für jedes Programm geschehen. Besser ist die Verwendung einer einzigen Skriptdatei, die alle gemeinsamen Befehle für das Linken jedes einzelnen Programms enthält. Dann kann der einzige Befehl (FILE) für jede Skriptdatei in der Kommandozeile über eine Batchdatei (BATFILE.BAT) angegeben werden. Der jeweilige Programmname kann dann über einen durch DOS ersetzbaren Parameter (z.B. %1) übergeben werden.

BATFILE.BAT:

```
COMPILE %1
BLINKER FILE %1 @Myapp
```

Die BATFILE.BAT kann dann zusammen mit der Angabe eines Programmnamens immer wieder über den DOS-Prompt aufgerufen werden.

```
BATFILE testapp
```

Dabei ist ‘testapp’ der Name des Programms, welches kompiliert und gelinkt werden soll. ‘BATFILE’ ist der Batchdatei-Aufruf.

Beispiel 2

Angenommen, ein Entwickler erstellt verschiedene Kopien eines Programms, um sie an Kunden zu verkaufen. Er möchte jeder Kopie des Programms, die er verkauft, eine individuelle Seriennummer zuweisen. Dazu kann er den Blinker-Befehl BLINKER EXECUTABLE SERIAL einsetzen, um einen 50 Zeichen langen String in die .EXE-Datei ‘einzubrennen’, der zum Beispiel Informationen über die Version, die Seriennummer, die Firma usw. enthalten kann. Zu diesem Zweck könnte der Befehl BLINKER EXECUTABLE SERIAL in die Skriptdatei eingesetzt werden, die Skriptdatei müßte dann jedoch vor jedem Linklauf neu editiert werden. Es wäre viel einfacher, diesen Befehl in einer Batch-Datei BATFILE.BAT zu plazieren und den Seriennummernstring der Batchdatei als Parameter zu übergeben:

BATFILE.BAT:

```
BLINKER @Myapp BLINKER EXECUTABLE SERIAL %1
```

Diese Batchdatei kann dann immer wieder neu mit der Angabe der jeweils gewünschten Seriennummer aufgerufen werden:

```
BATFILE 00005414
```

wobei ‘00005414’ die Seriennummer ist, die ‘eingebrannt’ werden soll. ‘BATFILE ist er Batchdatei-Aufruf.

Hier liegen unzählige Möglichkeiten, über Batch-Dateien Parameter an Blinker-Befehle zu übergeben.

Beispiel 3

Blinkerbefehle in einer Linkdatei können ergänzt werden durch Befehle auf der Kommandozeile. Falls ein Benutzer beispielsweise mit der Größe des Programm-Stacks experimentieren muß, um die optimale Eistellung für das Programm zu finden, kann die Zeile

```
STACK <nnnn>
```

in die Skriptdatei eingefügt werden, wobei <nnnn> den benötigten Wert darstellt. Aber das würde dazu führen, daß die Skriptdatei jedesmal editiert werden müßte, wenn der Wert sich ändert. Auch hier ist es einfacher, eine Batch-Datei einzusetzen:

BATFILE.BAT:

```
BLINKER @Myapp STACK %1
```

So sieht der Aufruf am DOS-Prompt aus:

```
BATFILE 5120
```

5120 ist der Wert für die Größe des Programm-Stacks. Wie Sie sehen, ist es eine einfache Angelegenheit, den Wert vom DOS-Prompt aus zu ändern. Wenn der passende Wert gefunden wurde, kann er in die Skriptdatei fest eigetragen werden.

Beinahe alle Blinker-Befehle können so manipuliert werden, wobei der Benutzer die Inhalte der Umgebungsvariablen, die Einstellungen zur Konfiguration der Speicherausnutzung kontinuierlich verändern und verschiedene Optionen ein- oder ausschalten kann.

Beispiel 4

Eine Skriptdatei kann in mehrere Dateien aufgeteilt werden. Jede davon kann in der Kommandozeile genau angegeben werden:

BATFILE.BAT:

```
BLINKER @Myapp1 STACK %1 @Myapp2
```

Sowohl Myapp1 als auch Myapp2 sind Skriptdateien, die durch die Teilung von MYAPP.LNK erzeugt wurden. Dies erlaubt die Plazierung des STACK-Befehls inmitten der BLINKER-Befehle für diesen Linklauf.

Link-Skript-Dateien

Verschachtelte Skriptdateien

Sie werden auf der Diskette einige Dateien mit der Erweiterung .LNK finden, auch Linkdateien genannt. Diese wurden beigefügt, um den Linklauf zu vereinfachen. Weiterhin wird damit ein besonderer Vorteil von Blinker, nämlich die Möglichkeit, Bibliotheken durch den Einsatz des MODULE-Befehls teilweise in den Overlay-Bereich mit hineinzunehmen, ausgenutzt.

Das hat den Effekt, daß einzelne .OBJ-Dateien aus ihren Bibliotheken gezogen und entweder in die Wurzel des Programmes (Root) oder in den Overlay-Bereich gelegt werden. Dies geschieht in Abhängigkeit von der Position des MODULE-Befehls. Auf diese Art kann jede verschachtelte Skriptdatei an das jeweilige Programm angepaßt werden. Dieses geschieht, indem die Zeilen, in denen Module aufgeführt sind, die nicht in den Root- bzw. Overlay-Bereich einbezogen werden sollen, als Kommentarzeilen gekennzeichnet werden.

Sie können auch Ihre eigenen verschachtelten Skriptdateien problemlos erzeugen und diese jeder Zeit überarbeiten.

Auf jede dieser Dateien können Ihre eigenen Skriptdateien Bezug nehmen, wenn dem Dateinamen ein '@' vorangestellt wird.

Als Beispiel überarbeiten wir unsere bisherige Muster-Skriptdatei, die wie folgt aussehen könnte:

```
MYAPP.LNK:

OUTPUT Myapp
STACK 4096
BLINKER EXECUTABLE EXTENDED
FILE Myapp1
FILE Myapp2
FILE Myapp3
```

```
@MSVC     # Verweis auf eine verschachtelte Linkdatei
```

In diesem Fall ist MSVC.LNK die verschachtelte Skriptdatei.

Sie könnte folgende Befehle enthalten:

```
MSVC.LNK:

LIB Mylib
LIB Alib
LIB Blib
# immer eine Ersatzbibliothek angeben
SEARCH blxmvc15
# (optionale) Compiler Spachbibliothek
LIB llibce
```

Dies erlaubt dem Entwickler, seine benötigten Routinen in einer separaten Skriptdatei abzulegen, die er mit allen seinen Programmen kombinieren kann.

Die Blinker-Hilfsprogramme

GPFs mit BLIGPF auffinden

Die Erzeugung von Import-Bibliotheken mit BLILIB

Die Blinker-Hilfsprogramme

GPFs mit BLIGPF auffinden

Das Hilfsprogramm BLIGPF liest die GPF-Ausgabe, die beim Auftreten einer Allgemeinen Schutzverletzung auf dem Bildschirm angezeigt wird, und verwendet die von Blinker zur Linkzeit erzeugte MAP-Datei zum exakten Auffinden der Stelle im Programm, die den GPF verursacht hat. Dies automatisiert das manuelle Analysieren, wie es im Abschnitt ‘Analyse von Allgemeinen Schutzverletzungen’ in Kapitel 5 beschrieben ist.

Als Ergebnis wird eine einfache Information (einschließlich des Symbolnamens) geliefert, die auf den Programmcode verweist, der ablief, als der GPF auftrat. Diese Information muß nicht unbedingt auf den fehlerverursachenden Code verweisen, da Allgemeine Schutzverletzungen auch durch die Übergabe fehlerhafter Parameter an richtigen Programmcode ausgelöst werden können.

Um BLIGPF zu verwenden, erzeugen Sie zuerst eine MAP-Datei über den Blinker Linkskript-Befehl MAP S,A und stellen Sie sicher, daß sich diese Datei im selben Verzeichnis befindet wie die .EXE- oder .DLL-Datei, die den Fehler verursacht. Führen Sie das Programm aus, bis der Fehler auftritt und starten Sie unmittelbar danach BLIGPF vom DOS-Prompt aus, um den GPF-Dump zu analysieren, der noch auf dem Bildschirm sichtbar ist. Falls sich die MAP-Datei nicht im aktuellen Verzeichnis befindet, kann sie BLIGPF nicht finden. In diesem Fall können Sie aber die MAP-Datei mit vollem Pfadnamen in der Kommandozeile hinter BLIGPF angeben.

Die Blinker-Hilfsprogramme

Die Erzeugung von Import-Bibliotheken mit BLILIB

Das Programm BLILIB liest eine Moduldefinitionsdatei, die EXPORT-Definitionen für eine DLL enthält, oder die aktuelle DLL-Datei selbst und erzeugt eine Moduldefinitionsdatei für eine Import-Bibliothek, in der die entsprechenden IMPORT-Definitionen für Progamme, die auf diese DLL zugreifen, zur Verfügung stehen.

Die Syntax der BLILIB-Befehlszeile ist wie folgt:

```
BLILIB /I{D | E} inputdatei /O{I | L} outputdatei [optionen]
```

/Ix zeigt die Eingabedatei an:

/ID inputdatei	Eine .DLL-Datei mit exportierten Namen
/IE inputdatei	Eine Moduldefinitionsdatei mit Export-Definitionen

/Ox zeigt die Ausgabedatei an:

/OI outputdatei	Eine Moduldefinitionsdatei mit Import-Definitionen
/OL outputdatei	Eine LIB-Datei mit Import-Definitionen

Ausführliche Informationen zur Verwendung von BLILIB und zu Importbibliotheken allgemein finden Sie im Abschnitt ‘Der BLILIB Bibliotheksmanager’ in Kapitel 6.

Make-Programme

Blinker funktionieret problemlos mit allen standard Make-Programmen wie MAKE, NMAKE, Opus MAKE, RMAKE etc.

Wenn Sie DOS-basierte Batch-Dateien oder Make-Programme unter Windows oder OS/2 ausführen, denken Sie daran, Blinker expliziet als BLINKER.COM anzugeben, damit er als DOS-Programm läuft. Falls ‘.COM’ nicht angegeben ist, könnte Windows den Blinker als ein Windows-Programm starten. Dadurch könnte dann ein Make-Programm mit der Ausführung fortfahren, ohne die Beendigung des Linkvorgangs durch Blinker abzuwarten.

Wenn Sie Interesse an einem leistungsfähigen Freeware Make-Programm haben, werfen Sie einen Blick auf PBMake im Internet unter ftp://ftp.iag.net/pub/clipper/utils/pbm214r.zip

DOS-Extender-Programm

Zum Linken von DOS-Extender- oder Dual Mode-Programmen muß immer eine zusätzliche Blinker-Bibliothek für den entsprechenden Compiler angegeben werden. Diese Bibliothek enthält .OBJ-Ersetzungen für alle in den Compiler-Bibliotheken enthaltenen .OBJs, die nicht im Protected Mode

verwendet werden können.

Die Namen aller Blinker-Bibliotheken besitzen die Form BLXcccnn, wobei ccc die Firma und den Compiler bezeichnet und nn die Versionsnummer des Compilers enthält. Die Bibliotheken sind alle im Large Modell kompiliert und sollten daher auch nur mit Large-und Huge-Modell-Programmen verwendet werden.

Die entsprechende Compiler-Bibliothek muß immer genau vor der standardmäßigen Compiler-Bibliothek oder vor der Skriptdatei, die die Compiler-Bibliothek enthält, über SEARCH angegeben werden.

Beispiel für Microsoft Visual C++:

```
SEARCH BLXMVC15  # Blinker-Bibliothek
LIB LLIBCE          # MSVC-Bibliothek
```

Beispiel für CA-Clipper 5.3:

```
SEARCH BLXCLP53  # Blinker-Bibliothek
LIB CLIPPER         # CA-Clipper-Bibliothek
```

oder:

```
SEARCH BLXCLP53  # Blinker-Bibliothek
@CL530MID        # Skriptdatei, die die CA-Clipper-
                 # Bibliothek enthält
```

Ist die jeweilige Compiler-Bibliothek nicht explizit erwähnt, dann geben Sie den Befehl SEARCH mit der Blinker-Bibliothek in der Linkdatei als letzte Bibliothek an.

Wenn Sie Drittanbieter-Bibliotheken (Third-Party-Produkte) verwenden, dann vergewissern Sie sich bitte, ob diese Bibliotheken im Protected Mode oder im Dual Mode eingesetzt werden können, bevor Sie versuchen, diese in ein DOS-Extender-Programm einzubauen. Blinkinc kann nicht dafür garantieren, daß alle Bibliotheken overlayfähig oder Protected Mode-kompatibel sind oder gar beides. Aktuelle Einzelheiten zu uns bekannten kompatiblen Bibliotheks-Versionen finden Sie in der Datei 3RDPARTY.TXT.

Drittanbieter-Bibliotheken müssen normalerweise vor dem SEARCH-Befehl der Blinker-Bibliothek angegeben werden, wenn für das Produkt keine anderweitigen Angaben gemacht wurden.

Weiter Informationen gibt es im Kapitel 3, ‘DOS Extended-/Dual Mode-Programme’.

Komprimierte .EXEs und .DLLs

Blinker unterstützt eine Komprimierung für Windows und DOS-Extended .EXE- und .DLL-Dateien sowie für Real-Mode DOS .EXE-Dateien. Diese Eigenschaft kann bei der endgültigen Version Ihrer .EXE- oder .DLL-Applikation vor der Auslieferung eingesetzt werden. Die Komprimierung reduziert Ihre Datei um 50% und mehr. Die komprimierten Dateien können ausgeliefert und ohne zusätzlichen Aufwand installiert werden, da sie sich (transparent für den Anwender) beim Programmstart automatisch dekomprimieren.

Der Komprimierungsalgorithmus ist äußerst effizient - liefert er doch eine Kompressionsrate von nur wenigen Prozent über der, die durch die derzeit besten Komprimierungsprogramme erzielt werden - besitzt jedoch den Vorteil einer transparenten Dekomprimierung auf Segmentbasis zur Ladezeit. Der Komprimierungsfaktor ist auf einer Skala von 1 bis 9 konfigurierbar, wobei 1 die längste Zeit zur Komprimierung benötigt, jedoch auch die beste Rate liefert, und 9 für die schnellsten Komprimierung und Dekomprimierung steht.

Um die Komprimierung einzuschalten, fügen Sie einfach den Befehl BLINKER EXECUTABLE COMPRESS <faktor> hinter die Befehle, die den .EXE-Typ bestimmen (wie z.B. BLINKER EXECUTABLE EXTENDED), in die Link-Skriptdatei ein.

Sehen Sie hierzu auch den Abschnitt ‘.EXE und .DLL Dateikomprimierung’ im Kapitel 5, ‘Besondere Merkmale’, für weitere Einzelheiten.

Schnelleinstieg für verschiedene Compiler

Wenden Sie sich bitte an die Onlinehilfe für detaillierte Anweisungen, wie Sie möglichst bald mit Blinker und Ihrem ausgewählten Compiler starten können. Nicht für alle Compiler werden Beispiele zum Linken von der Kommandozeile aus gezeigt, weil auch das einfachste Programm eine Link-Skriptdatei benötigt. Wollen Sie keine Link-Skriptdatei verwenden, dann können Sie sich weiter oben in diesem Kapitel einige allgemeine Beispiele zum Linken über die Kommandozeile ansehen.

Für jeden Compiler wurde ein Beispielprogramm mit der entsprechenden Assembler- oder Compiler-Kommandozeile veröffentlicht. Ebenfalls angegeben sind die Optionen für DOS- oder DOS-Extender-Programme. Um den Quelltext auch unter Windows einsetzen zu können, sind möglicherweise einige Veränderungen erforderlich. Details zur Windows-Programmierung bzw. zu alternativen oder zusätzlich benötigten Optionen finden Sie in der jeweiligen Compiler- oder Assembler-Dokumentation.

Jeder Compiler-Abschnitt enthält bis zu drei Link-Skriptdateibeispiele, soweit möglich: ein DOS-Extended-Programm, ein Dual Mode- Programm oder ein Programm mit dynamischen Overlays und ein Windows-Programm. Jedes Programm ist vollständig kommentiert, um die Verwendung der verschiedenen Befehle zu erläutern. Weitere Informationen über die einzelnen Befehle finden Sie in der Online-Hilfe.

Ein DOS-Extended-Programm benötigt die einfachste Link-datei. Sie enthält normalerweise eine Liste von Eingabedateien, den Namen der Ausgabedatei, die Blinker-Extender-Bibliothek für den entsprechenden Compiler und ein oder zwei zusätzliche Blinker-Befehle. Eine DOS-Extender-Linkdatei kann durch das Löschen des Befehls BLINKER EXECUTABLE EXTENDED so umgewandelt werden, daß mit dieser Datei ein Programm ohne Overlays erzeugt werden kann.

Für ein Dual Mode- oder ein Programm mit dynamischen Overlays sind etwas kompliziertere Skripts erforderlich, da die Module für die Overlay-Erzeugung zwischen einem BEGINAREA und einem ENDAREA stehen müssen. Die Overlays werden benötigt, wenn im Real Mode gearbeitet wird. Eine Dual Mode-Linkdatei kann durch das Löschen des Befehls BLINKER EXECUTABLE DUAL so umgewandelt werden, daß mit dieser Datei ein DOS-Programm mit dynamischen Overlays erzeugt wird.

Ein Windows-Programm (falls es vom Compiler unterstützt wird) verwendet den Befehl DEFFILE zur Angabe einer Moduldefinitionsdatei. Diese wird ebenfalls angezeigt. Alternativ dazu können die Befehle aus der Definitionsdatei auch in der Link-Skriptdatei zwischen den Befehlen DEFBEGIN and DEFEND angegeben werden.

Compiler und Sprachen mischen

Blinker kann .OBJ- und .LIB-Dateien von fast jedem Compiler kombinieren, jedoch werden dadurch die Beschränkungen, die jeder Compiler bezüglich gemischten Programmiersprachen hat, nicht aufgehoben. CA-Clipper und BASIC z.B. können nicht zu einer Anwendung kombiniert werden. Jede dieser Sprachen erwartet zur Laufzeit ihre eigenen Routinen für Start, Umgebungsinitialisierung und Speicherverwaltung.

Wenn man Sprachen mischt, müssen außerdem Anforderungen in Bezug auf Aufrufkonvention, Parameterübergabe und Linkregeln erfüllt sein. Überprüfen Sie die Dokumentation jedes Compilers auf solche Anforderungen hin, bevor Sie linken.

Alaska Xbase++

Borland Turbo Assembler

Borland Turbo C / Turbo C++

- Borland C++
- CA-Clipper Sommer 87
- CA-Clipper 5.x
- FORCE
- Microsoft Assembler
- Microsoft BASIC PDS 7.1 / VB DOS 1.0
- Microsoft QuickBASIC 4.5
- Microsoft C / C++ / Quick C
- Microsoft FORTRAN
- Microsoft PASCAL
- Microsoft Visual C++
- Stony Brook Modula 2
- Stony Brook Pascal+
- Symantec C++
- WATCOM C/C++
- Zinc Application Framework für DOS

Schnelleinstieg für verschiedene Compiler

Alaska Xbase++

Blinker unterstützt Alaska Xbase++ 1.1 Service Release 1 (1.1 SL1) und später zur Erzeugung von 32 Bit Windows Programmen, einchließlich automatischer Ausführung von Blinker durch PBUILD und voller Unterstützung des Xbase++ Debuggers.

Stellen Sie bitte sicher, daß Sie mindestens die Version 1.1 mit Service Release 1 vom Juni 1998 verwenden, bevor Sie sich wegen Support an uns oder Alska wenden.Falls Sie eine ältere Version von Xbase++ mit Blinker verwenden, können sich daraus die verschiedensten Fehler ergeben, wie z.B. GPFs zur Linkzeit, falsche Anzeige der Programm Quelltexte im Debugger oder ein Blinker Fehler 1165 bei einer Import-Bibliothek für eine .DLL-Datei.

Wenn Sie ein Problem mit einem mit Blinker gelinkten Xbase++ Programm haben, versuchen Sie bitte immer zunächst, dasselbe Programm mit Alink zu linken und in derselben Umgebung auszuführen, bevor Sie sich wegen Unterstützuing an Blinkinc wenden. Denn wenn das Problem auch noch nach dem Linken mit Alink auftritt, dann liegt es offensichtlich nicht am Blinker und kann dann auch nicht von Blinkinc gelöst werden.

Setup

Defaultmäßig verwendet Xbase++ .OBJ-Dateien im COFF Format , das zur Zeit von Blinker nicht unterstützt wird.

Deshalb müssen Xbase++ und PBUILD so konfiguriert werden, daß das OMF Format für .OBJ- und .LIB-Dateien benutzt wird. Wenn Sie zwischen COFF- und OMF-Format umschalten, vergessen Sie nicht, mit PBUILD das Projekt komplett neu zu erzeugen, z.B. mit PBUILD /A.

Bevor Sie versuchen, Blinker mit Xbase++ zu verwenden, müssen Sie daher folgende Änderungen in Ihrer Entwicklungsumgebung durchführen:

Setzen Sie die DOS Umgebungsvariable LIB so, daß sie zuerst auf das OMF Unterverzeichnis des Xbase++ LIB Unterverzeichnisses (z.B. XPPW32\LIB\OMF) zeigt und erst dann auf das Xbase++ LIB Unterverzeichnis. Dadurch wird sichergestellt, daß Blinker die Bibliotheken im OMF-Format auswählt, anstatt die im COFF-Format. Andernfalls erhalten Sie einen Blinkerfehler 1165 zur Linkzeit.

Wenn Sie PBUILD verwenden, machen Sie folgende Änderungen oder Ergänzungen in jeder Ihrer .XPJ (PBUILD Project) Dateien:

```
OBJ_FORMAT = OMF
OMF_LINKER = BLINKER.COM /QM
```

(Falls Sie aus irgendeinem Grunde noch eine ältere Version von Xbase++ verwenden, müssen Sie diese alternativen Einstellungen vornehmen:

```
COMPILE_FLAGS = -OMF
LINKER = BLINKER.COM /QM
```

Aber dies wird nicht offiziell unterstützt, und das resultierende Programm funktioniert möglicherweise nicht immer korrekt.)

Batch- und Make-Dateien

Wenn Sie Blinker aus einer Batch- oder Make-Datei aufrufen, müssen Sie ein Linkskript wie unten gezeigt erstellen. Außerdem muß Blinker als BLINKER.COM aufgerufen werden, damit er als DOS-

Programm startet. Weiterhin müssen alle .PRG-Dateien im OMF-Format übersetzt werden durch Angabe von –OMF als Befehlszeilenparamter von XPP.

Xbase++ Linkskripte bestehen typischerweise aus einer Liste von .OBJ-Dateien gefolgt von einigen einfachen Blinkerbefehlen zur Bestimmung des Typs der Ausgabedatei.

Wenn die Ausgabedatei Informationen für den Debugger enthalten soll, verwenden Sie den CODEVIEW Linkbefehl. Weitere Details finden Sie in der Online-Hilfe zu den Befehlen CODEVIEW, CVON und CVOFF.

Beispiel zum Linken von TEST.PRG

```
# TEST.LNK
CODEVIEW         # Informationen für den Debugger
FILE TEST        # TEST.OBJ Datei
OUTPUT TEST      # Dateiname für Ausgabe
NOBELL           # Kein Piepsen zum Schluß
```

Um den Typ der ausführbaren Datei (32 Bit) zu bestimmen, fügen Sie die folgenden 4 Zeilen an das Ende des Skripts hinzu:

```
DEFBEGIN
   NAME TEST
   EXETYPE NT
DEFEND
```

NAME TEST bedeutet, daß eine EXE erzeugt wird und daß der Modulname des Programmes TEST ist. Um eine DLL mit demselben Namen zu erzeugen, erstzen Sie NAME TEST einfach durch LIBRARY TEST. Weitere Einzelheiten zu NAME und LIBRARY finden Sie in der Online-Hilfe unter 'Befehle zur Moduldefinition'.

Wenn Sie ein Windows GUI Programm erzeugen wollen, reicht der Befehl EXETYPE NT aus. Aber wenn Sie bestehende Clipper-Programme konvertieren oder neue Character-Mode-Programme schreiben, muß das EXETYPE SUBSYSTEM angegeben werden als CONSOLE.

```
. . .
   EXETYPE NT SUBSYSTEM CONSOLE
. . .
```

In der Online-Hilfe zu EXETYPE unter 'Befehle zur Moduldefinition ' finden Sie weitere Informationen zu den alternativen Subsystemen.

Beispielprogramme

Wenn Sie bei der Benutzung von Blinker mit Xbase++ einen Link- oder Runtimefehler erhalten, versuchen Sie bitte immer, ein kleines Testprogramm zu erzeugen, das dieselbe Aufgabe ausführt. Denn die meisten Probleme lassen sich klären, sobald man sie in einem kleinen Testprogramm isoliert hat.

Das XPP Unterverzeichnis des Blinker Installationsverzeichnisses enthält Beispielprogramme und Linkskript-Dateien für den Gebrauch von Blinker mit Xbase++. Enthalten sind auch ein Beispiel für ein kleines Programm, das eine separate .DLL verwendet, sowie DBU.LNK zum Linken des CA-Clipper DBU-Programms.

Beachten Sie bitte, daß der CA-Clipper Sourcecode von DBU eventuell angepaßt werden muß zur Kompilierung mit Xbase++. Blinkinc bietet keine Unterstützung an für diese Anpassungen. Die DBU.LNK Datei wird also nur so angeboten, wie sie ist , und sie dient lediglich der Information.

Schnelleinstieg für verschiedene Compiler

Borland Turbo Assembler

Blinker unterstützt alle Versionen des Turbo Assembler. Die folgenden in Turbo Assembler geschriebenen Beispielprogramme sollen Ihnen in diesem Abschnitt die Grundlagen des Linkens von Turbo Assembler-Programmen mit Blinker veranschaulichen.

DOS-Extender-Programme, Dual Mode-Programme und Programme, die ins Overlay gelegt werden, müssen Far-Aufrufe nutzen, d.h. mit dem Medium, Large oder Huge Speichermodell assembliert werden. DOS-Extender und Dual Mode unterstützen das Medium Modell nicht.

TEST.ASM:

```
.model large
.stack 100h
; Prozeduren für Overlay müssen als far deklariert werden
extrn     limerick:far
.code
proc      main      far
          mov       ax,DGROUP
          mov       ds,ax
          call      limerick
          mov       ax,4c00h
          int       21h
endp      main
end       main
```

LIMERICK.ASM:

```
.model    large
public    limerick
.data
testmsg   db           'Es war ein junger Mitarbeiter von Borland'
          db        0dh,0ah,'$'
.code
; Prozeduren für Overlay müssen als far deklariert werden
proc      limerick  far
          mov       ah,9
          mov       dx,offset DGROUP:testmsg
          int       21h
          ret
endp      limerick
end
```

Diese beiden Module werden folgendermaßen assembliert:

```
TASM test
TASM limerick
```

DOS-Extender- / Programm ohne Overlays

```
# TEST.LNK (Das ist ein Kommentar)
# Für DOS-Real-Mode-Programm: Nächste Zeile auskommentieren
BLINKER EXECUTABLE EXTENDED   # Erzeugt ein DOS-Extender-Programm
FILE test
FILE limerick
```

Dadurch linkt Blinker die beiden in den FILE-Befehlen angegebenen Objektmodule (test.obj und

limerick.obj) zu einem DOS-Extender-Programm. Das Programm bekommt den Namen TEST.EXE, weil das der Name der ersten .OBJ-Datei in einem FILE-Befehl ist.

Dual Mode- / Programm mit Overlays

Beachten Sie bitte, daß das Startmodul in einem Assembler-Programm nicht overlayfähig ist.

```
# TESTDUAL.LNK
# Nächste Zeile nur für dynamische Overlays auskommentieren
BLINKER EXECUTABLE DUAL    # Erzeugt Dual Mode-Programm
BLINKER OVERLAY OPSIZE 50      # 50 KB Overlay-Bereich
BLINKER OVERLAY PAGEFRAME ON  # Overlays laufen im Pageframe
BLINKER MESSAGE WINK           # Nur mit einem Auge blinken
FILE test                  # Diese Datei nicht im Overlay
BEGINAREA                  # Start von Overlays
FILE limerick              # Diese Datei im Overlay
ENDAREA                        # Ende der Overlays
OUTPUT t                   # Name der Ausgabe.EXE
MAP A,S                        # Erzeugt eine MAP-Datei
```

16 Bit Windows-Programm

```
#TESTWIN.LNK
FILE test
FILE limerick
DEFFILE TEST     # TEST.DEF verwenden

#TEST.DEF
NAME             TEST
DESCRIPTION          'Ein kleines Windows-Programm'
EXETYPE              WINDOWS 3.1
HEAPSIZE         1024
STACKSIZE        8192
```

Schnelleinstieg für verschiedene Compiler

Borland Turbo C / Turbo C++

BLINKER unterstützt alle Versionen von Turbo C und Turbo C++ zur Erzeugung von DOS Realmode Programmen. Die folgenden in Turbo C oder Turbo C++ geschriebenen Beispielprogramme werden während dieses Abschnittes verwendet, um Ihnen die Grundlagen des Linkens von Turbo C- und Turbo C++ Programmen mit Blinker zu veranschaulichen.

Turbo C++ benötigt das Link-Skript, um in FILE das Startmodul C0x.OBJ als erste Datei sowie optional die Bibliotheken EMU.LIB, MATHx.LIB als Bibliotheken in LIB und Cx.LIB und GRAPHICS.LIB als letzte Bibliotheken in LIB einfügen zu können. x kennzeichnet dabei das Speichermodell.

Programme, die ins Overlay gelegt werden, müssen Far-Aufrufe nutzen, d.h. mit dem Medium, Large oder Huge Speichermodell assembliert werden.

Hinweis: Die Stackgröße eines Turbo C / Turbo C++ Programms muß innerhalb des Programmcodes durch die Variable _stklen gesetzt werden, nicht mit dem STACK-Linkbefehl (z.B. extern unsigned _stklen = 8192;). Bitte suchen Sie in der Borland Dokumentation nach weiteren Einzelheiten.

TEST.C:

```
void limerick (void);

void main (void)
{
   limerick ();
}
```

LIMERICK.C:

```
#include <stdio.h>
void limerick (void)
{
   printf (" Es war ein Knabe von Borland\n");
}
```

Diese beiden Module werden wie folgt als ‘Large’ Modell-Code kompiliert:

```
TCC -c -ml test.c
TCC -c -ml limerick.c
```

Programm ohne Overlays

```
# TEST.LNK (Das ist ein Kommentar)
FILE c0l            # Startroutine für Turbo C
FILE test           # das Hauptprogramm
FILE limerick
LIB cl              # Turbo C-Bibliothek
OUTPUT t            # Name der Ausgabe.EXE
MIXCASE               # Groß-/Kleinschreibung unterscheiden
```

Dadurch linkt Blinker die beiden in den FILE-Befehlen angegebenen Objektmodule (test.obj und limerick.obj) zu einem Programm mit dem Namen TEST.EXE, so wie im OUTPUT-Befehl angegeben.

Overlay-Programm

```
# TESTOVL.LNK
BLINKER OVERLAY OPSIZE 50     # 50 KB Overlay-Bereich
```

```
BLINKER OVERLAY PAGEFRAME ON  # Overlays laufen in Pageframe
BLINKER MESSAGE WINK          # Nur mit einem Auge blinken
FILE c0l                   # Startroutine für Turbo C
FILE test                  # Diese Datei nicht im Overlay
BEGINAREA                  # Overlay-Start
   FILE limerick           # Diese Datei im Overlay
ENDAREA                       # Overlay-Ende
LIB cl                     # Turbo C-Bibliothek, nicht
                           # overlayfähig
OUTPUT t                   # Name der Ausgabe.EXE
MAP A,S                       # Erzeugt eine MAP-Datei
```

Schnelleinstieg für verschiedene Compiler

Borland C++

BLINKER unterstützt alle Versionen von Borland C++ für DOS-Realmode- und Windowsprogramme, und ab Version 3.1 auch Protected Mode Programme. Die folgenden, in Borland C++ geschriebenen Beispielprogramme, werden während dieses Abschnittes verwendet, um Ihnen die Grundlagen des Linkes von Borland C++Programmen mit Blinker zu veranschaulichen.

Borland C++ benötigt im Link-Skript das Startmodul C0x.OBJ (bzw. C0xPnn aus dem OBJ Unterverzeichnis von Blinker für Protected Mode) als erste Datei in FILE sowie optional die Bibliotheken EMU.LIB, MATHx.LIB als Bibliotheken in LIB und Cx.LIB und GRAPHICS.LIB als letzte Bibliotheken in LIB. x kennzeichnet dabei das Speichermodell. (GRAPHICS.LIB wird zur Zeit in Protected Mode Programmen nicht unterstützt).

Benutzen Sie Borlands globale Variable _maxfarrealloc, um die Anzahl der Selektoren zu verringern, die für jedes farmalloc() Segment eines DOS-Extender Programmes belegt werden.

DOS-Extender-Programme, Dual Mode-Programme und Programme, die ins Overlay gelegt werden, müssen Far-Aufrufe nutzen, d.h. mit dem Medium, Large oder Huge Speichermodell assembliert werden. DOS-Extender und Dual Mode unterstützen das Medium Modell nicht.

Hinweis: Die Stackgröße eines Borland C++ Programms muß innerhalb des Programmcodes durch die Variable _stklen gesetzt werden, nicht mit dem STACK-Linkbefehl (z.B. extern unsigned _stklen = 8192;). Bitte suchen Sie in der Borland Dokumentation nach weiteren Einzelheiten.

Das OBJ Unterverzeichnis enthält die Dateien FP7BCPnn.OBJ. Diese müssen zu allen Protected Mode Programmen gelinkt werden, die LIB FP87 anstelle von LIB EMU verwenden, um den Gebrauch des mathematischen Coprozessors für Floating Point Operations zu erzwingen. FP7BCPnn.OBJ sollte die zweite .OBJ-Datei im Linkskript sein, d.h. direkt hinter FILE C0xPnn.OBJ stehen.

Für Windowsprogramme kann es erforderlich sein, die Bibliothek IMPORT.LIB explizit mit SEARCH aufzurufen, um nicht aufgelöste Referenzen zu beseitigen. Fügen Sie einfach die Zeile

```
SEARCH IMPORT
```

an den Anfang der Bibliotheksliste, falls Sie ‘unresolved externals’ bei Windows API Funktionen erhalten.

TEST.CPP:

```
void limerick (void);
void main (void)
{
   limerick ();
}
```

LIMERICK.CPP:

```
#include <stdio.h>
void limerick (void)
{
   printf ("Es war ein Mitarbeiter von Borland\n");
}
```

Die beiden Module werden folgendermaßen als Large Model-Code kompiliert:

```
BCC -c -ml test.cpp
BCC -c -ml limerick.cpp
```

DOS-Extender-Programme / Programm ohne Overlays

```
# TEST.LNK (Das ist ein Kommentar)
# Nur für dynamische Overlays auskommentieren:
BLINKER EXECUTABLE EXTENDED   # Erzeugt ein DOS-Extender-
                              # Programm
FILE c0lpnn                      # Borland C++ Startcode
FILE test
FILE limerick
SEARCH blxbcpnn               # Blinker-Dual-Mode-Bibliothek
LIB cl                        # Borland C++ Bibliothek
OUTPUT t                      # Name der Ausgabe.EXE
MIXCASE                          # Groß-/Kleinschreibung unterscheiden
```

Dadurch linkt Blinker die beiden in den FILE-Befehlen angegebenen Objektmodule (test.obj und limerick.obj) zu einem DOS-Extender-Programm mit dem Namen t.EXE, wie in dem OUTPUT Befehl angegeben.

Dual Mode-Programm / Overlay-Programm

```
# TESTDUAL.LNK
# Nächste Zeile auskommentieren, falls nur dynamische Overlays
# benötigt werden
BLINKER EXECUTABLE DUAL    # Erzeugt Dual Mode-Programm
BLINKER OVERLAY OPSIZE 50     # 50 KB Overlay-Bereich
BLINKER OVERLAY PAGEFRAME ON  # Overlays in Pageframe
BLINKER MESSAGE WINK          # Nur mit einem Auge blinken
FILE c0lpnn                   # Borland C++ Startcode
FILE test                  # Diese Datei nicht im Overlay
BEGINAREA                  # Overlay-Start
   FILE limerick           # Diese Datei im Overlay
ENDAREA                       # Overlay-Ende
SEARCH blxbcpnn            # Blinker-Dual Mode-Bibliothek
LIB emu,mathl,cl           # Borland C++ Bibliotheken sind
                           # nicht overlayfähig
LIB graphics               # NICHT unterstützt im Protected Mode
OUTPUT t                   # Name der Ausgabe.EXE
MAP A,S                       # Erzeugt eine MAP-Datei
MIXCASE                       # Groß-/Kleinschreibung unterscheiden
```

16-Bit Windows-Programm

```
# TESTWIN.LNK
FILE c0wl        # Borland C++ Startcode
FILE test
FILE limerick
LIB cwl          # Borland C++ Bibliothek
MIXCASE             # Groß-/Kleinschreibung unterscheiden
DEFFILE TEST     # Verwendet TEST.DEF

; TEST.DEF
NAME             TEST
DESCRIPTION          'Kleines Windows-Programm'
EXETYPE          WINDOWS 3.1
HEAPSIZE         1024
STACKSIZE        8192
```

Schnelleinstieg für verschiedene Compiler

CA-Clipper Sommer ’87

Blinker unterstützt CA-Clipper Sommer ’87 zwar, kann aber für diesen Compiler nur overlayfähige DOS-Pogramme erzeugen. Die folgenden in Clipper Sommer ’87 geschriebenen Beispielprogramme werden während dieses Abschnittes verwendet, um Ihnen die Grundlagen des Linkprozesses von CA-Clipper Sommer ’87-Programmen mit Blinker zu veranschaulichen.

TEST.PRG:

```
DO limerick
QUIT
```

LIMERICK.PRG:

```
? "Da war ein junger Mitarbeiter von Nantucket"
RETURN
```

Die beiden Module werden folgendermaßen kompiliert:

```
CLIPPER test -m
CLIPPER limerick -m
```

Um den Blinker von der Kommandozeile aus zu starten:

```
BLINKER FILE test, limerick LIB extend, clipper
```

Das veranlaßt Blinker, die beiden Objektmodule, die nach dem FILE-Befehl angegeben wurden (test.obj und limerick.obj), und die beiden Bibliotheken (extend.LIB und clipper.LIB), die nach dem LIB-Befehl angegeben wurden, zu linken. Das Ergebnis ist ein ausführbares Programm mit dem Namen TEST.EXE, dem Name des ersten Objekt-Moduls, das nach dem FILE-Befehl genannt wurde.

Das inkrementelle Linken ist für den Blinker standardmäßig aktiviert, um die Entwicklungsarbeit zu beschleunigen.

Der Einsatz des Blinker von der Kommandozeile aus ist bei kleinen Programmen mit wenigen Befehlen günstig. Bei großen Programmen jedoch wird es nervtötend und schließlich völlig unmöglich, wenn die Anzahl der Befehle die Länge der Kommandozeile auf DOS-Ebene überschreitet. Wie die meisten Linkern kann auch der Blinker mit einer Skriptdatei, einer Textdatei mit einer Liste von Blinker-Befehlen, gesteuert werden.

Einsatz einer Link-Skriptdatei (Entwicklung, keine Overlays)

Die folgende Skriptdatei liefert das gleiche Ergebnis, wie die eben geschilderte Ausführung des Blinkers von der Kommandozeile aus:

```
# TEST.LNK (dies ist ein Kommentar, wie am #
# zu erkennen ist)
FILE test
FILE limerick
LIB extend, clipper     #CA-Clipper-Bibliotheken
```

HINWEIS: Die erste angegebene Datei muß die Clipper-Hauptprozedur der Anwendung enthalten. Um Blinker zu veranlassen, diese Skriptdatei auszuführen, geben Sie bitte folgenden Befehl ein:

```
BLINKER @test
```

Am '@'-Zeichen erkennt Blinker, daß nun der Name einer Skriptdatei folgt. Blinker nimmt standardmäßig die Dateierweiterung .LNK an. Es kann aber auch eine andere in der Kommandozeile angegeben werden.

Entwicklungsversion mit Overlays

```
# TESTOVL.LNK:
BEGINAREA           # Beginn des Overlay-Bereichs
   FILE test        # Gesamter Sommer '87-Code
   FILE limerick       # ist overlayfähig
   LIB extend          # EXTEND-Bibliothek ins Overlay
ENDAREA                # Ende des Overlay-Bereichs
LIB clipper
```

Beachten Sie, daß der gesamte Clipper-Code und die EXTEND.LIB in das Overlay gelegt werden können. Die folgende Zeile veranlaßt Blinker, die obenstehende Skriptdatei auszuführen:

```
BLINKER @testovl
```

Overlay-Programm, Endversion

```
# TESTS87.LNK
# Kommentare beginnen mit einem #-Zeichen
BLINKER INCREMENTAL OFF# Inkrementelles Linken aus
# Umgebungsvariable CLIPPER einbrennen
BLINKER EXECUTABLE CLIPPER F41;E0;R32;V15
BLINKER PROCEDURE DEPTH 50    # Setzen eines geeigneten Stack
BLINKER MEMORY PACK 10     # Speicherdefragmentierung aktiv
BLINKER OVERLAY OPSIZE 40  # Größe des Overlay-Pools setzen
OUTPUT test.exe         # Name der Ausgabe.EXE
BEGINAREA               # Start des Overlay-Bereichs
   FILE test               # Diese Datei ist overlayfähig
   FILE limerick           # Diese ebenfalls
   LIB extend              # Diese Bibliothek im Overlay abgelegt
ENDAREA                    # Ende des Overlay-Bereichs
LIB clipper
```

Die obengenannten Blinker-Befehle können überall in der Skriptdatei plaziert werden.

Der wichtigste dieser Befehle ist BLINKER INCREMENTAL OFF. Das inkrementelle Linken ist für den Blinker standardmäßig aktiviert, um die Entwicklungsarbeit zu beschleunigen.

Vor der Produktion der Endversion muß das inkrementelle Linken unbedingt ausgeschaltet werden. Inkrementelles Linken reduziert ebenso wie das Linken mit Debugger-Informationen den verfügbaren Speicher sowie die Ausführungsgeschwindigkeit einer Anwendung und ist nur für die Entwicklungsphase gedacht.

Während der Entwicklung einer Applikation ist diese Einstellung optimal. Sie sollten sie aber nicht

benutzen, wenn die Entwicklung abgeschlossen ist und eine Endversion erstellt werden soll. Auch bei auftretenden Speicherplatzproblemen müssen Sie auf das inkrementelle Linken verzichten.

Vergleiche auch den Abschnitt "Gebrauch des Blinker unter CA-Clipper" in der Online-Hilfe.

Schnelleinstieg für verschiedene Compiler

CA-Clipper 5.x

Blinker unterstützt CA-Clipper in den Versionen 5.01a, 5.2e und 5.3b bei der Erzeugung von DOS- und DOS-Extender-Programmen. Andere Versionen werden nicht unterstützt. Deshalb besorgen Sie sich bitte die entsprechenden Upgrades von Computer Associates, bevor Sie Blinker mit CA-Clipper 5.x einsetzen.

Die folgenden in CA-Clipper 5.x geschriebenen Beispielprogramme veranschaulichen Ihnen in diesem Abschnitt die Grundlagen des Linkens von CA-Clipper 5.x mit Blinker.

CA-Clipper 5.3 Workbench Link Templates für Real Mode, Dual Mode und Extended Mode Programs finden Sie in dem Blinker CLP\TPL Unterverzeichnis in den Dateien BLIRMD.TPL, BLIDUA.TPL und BLIEXT.TPL. Beachten Sie, daß incrementelles Linken in diesen Templates standardmäßig auf ON steht.

Weiterhin kann Blinker zusammen mit den Produkten Clip-4-Win, FiveWin oder CLProfiW und CA-Clipper 5.x Windowsprogramme erzeugen. Bitte verwenden Sie die Dokumentationen dieser Produkte für weitere Hinweise zum Einsatz mit Blinker.

Im OBJ Unterverzeichnis von Blinker befindet sich die Datei MPAR.OBJ, welche Sie mit allen CA-Clipper Windowsprogramen linken sollten. Diese Datei fixt ein Bug in CLIPPER.LIB Version 5.3b und früher, das ein Laden eines Windowsprogramms verhindert, falls eine bestimmte interne Tabelle eine 32 KB Grenze überschreitet. Dies konnte in einem CA-Clipper Windowsprogramm jeglicher Größe auftreten. Windows würde dann die hilfreiche Meldung 'Ein mit dem System verbundenes Gerät antwortet nicht' ausgeben und das Programm beenden.

TEST.PRG:

```
DO limerick
QUIT
```

LIMERICK.PRG:

```
? "Es war ein alter Mitarbeiter von CA"
RETURN
```

Die beiden Module werden folgendermaßen kompiliert:

```
CLIPPER test -m
CLIPPER limerick -m
```

Im folgenden wird der Ausdruck ‘Entwicklungsversion’ verwendet, wenn von einem inkrementell gelinkten Programm die Rede ist, und ‘Endversion’, wenn die Incremental-Option deaktiviert ist, also die Komprimierung der Symboltabelle und das Paging-System von CA-Clipper aktiviert wurden.

Der CA-Clipper Compiler 5.x fügt die Namen aller erforderlichen CA-Clipper-Bibliotheken in den Header aller Objektmodule ein, um die Suche nach den Standard-Bibliotheken zu unterstützen.

Zum Starten des Blinker von der Kommandozeile:

```
BLINKER FILE test, limerick
```

Dies veranlaßt Blinker, die beiden Objektmodule, die im FILE-Befehl angegeben wurden (test.obj und limerick.obj), zu linken. Das Ergebnis ist ein ausführbares Programm mit dem Namen test.exe. Das ist der Name des ersten Objektmoduls, das nach dem FILE-Befehl genannt wurde. Die CA-Bibliotheken werden automatisch mit eingebunden. Dabei wird angenommen, daß die LIB-Variable auf das Verzeichnis verweist, wo diese Bibliotheken abgelegt wurden.

Das inkrementelle Linken ist bei Blinker standardmäßig aktiviert, um die Entwicklungsarbeit zu beschleunigen.

Zwecks Speichereffizienz und für das abschließende Linken können Sie den gesamten CA-Clipper-Code über einem zusätzlichen Befehl ein einem einzigen Overlay ablegen:

```
BLINKER FILE test, limerick BLINKER INCREMENTAL OFF
```

In der Version 5.x wird der gesamte Clipper-Code unter Ausnutzung des Code Paging Systems von CA-Clipper in ein Overlay gelegt, wenn das inkrementelle Linken deaktiviert wird.

Wenn zusammen mit den Befehlen BLINKER EXECUTABLE EXTENDED bzw. BLINKER EXECUTABLE DUAL keine BEGINAREA und ENDAREA-Befehle verwendet werden, wird das Paging-System von CA-Clipper automatisch ausgeschaltet.

Im Falle von DUAL-Mode-Programmen wird auf diese Weise ein Programm erzeugt, das keine Overlays verwendet, also auch keine Overlay-Dateien zur Laufzeit benötigt. Es läuft natürlich weiterhin sowohl im Protected und im Real Mode. Dies lässt sich für .EXE-Dateien verwenden, die nicht auf Diskette zugreifen sollen, um dort Overlay-Dateien zu suchen, wie z.B. Installationsprogramme.

Im Falle von reinen EXTENDED-Programmen kann auf diese Weise in einigen Fällen ein besseres Laufzeitverhalten erzielt werden.

Protected Mode CA-Clipperprogramme zeigen einen Call Trace, wenn ein GPF auftritt NACHDEM das Hauptprogramm gestartet ist, d.h. nicht während der INIT Funktionen. Dieser Trace (Aufrufliste) ist so sortiert, das der letzte Funktionsaufruf als letztes angezeigt wird.

Der nächste Schritt ist die Erzeugung einer Skriptdatei.

Einfaches Programm in der Entwicklungsphase

Die folgende Skriptdatei entspricht dem oben angeführten ersten Beispiel:

```
TEST.LNK:

FILE test
FILE limerick
```

Noch einmal zur Erinnerung: Die CA-Clippper-Bibliotheken werden unter CA-Clipper 5.x automatisch mit eingebunden, vorausgesetzt, die LIB-Variable verweist auf das entsprechende Verzeichnis.

HINWEIS: Die erstgenannte Datei muß das CA-Clipper-Hauptprogramm der Anwendung enthalten.

Um die o.g. Skriptdatei mit Blinker auszuführen, benutzen Sie das ‘@’-Symbol, gefolgt von dem Namen der Skriptdatei (die Standarddateierweiterung lautet .LNK).

```
BLINKER @test
```

Einfaches Programm (Endversion)

Die folgende Skriptdatei entspricht dem zweiten Beispiel (siehe oben).

```
TEST2.LNK:

BLINKER INCREMENTAL OFF
FILE test
```

```
FILE limerick
```

Um diese Skriptdatei durch Blinker auszuführen:

```
BLINKER @test2
```

Der nächste Schritt ist die Erzeugung einer Skriptdatei zur Aufnahme zahlreicher Dateien, Bibliotheken und Blinker- Linkbefehle.

Overlay-Programm (Endversion)

```
# TESTC5X.LNK
# Kommentare beginnen mit dem #
# Inkrementelles Linken zugunsten optimaler
# Speicherausnutzung deaktivieren
BLINKER INCREMENTAL OFF
# Einbrennen der Clipper-Umgebungsvariablen
BLINKER EXECUTABLE CLIPPER F:41
BLINKER PROCEDURE DEPTH 50 # Setzen der Stackgröße
BLINKER OVERLAY OPSIZE 50  # Overlay-Pools auf 50 KB setzen
OUTPUT test.exe
BEGINAREA              # Beginn des Overlay-Bereichs
   FILE  test
   FILE  limerick
   LIB c:\test\mylib      # Bibliothek mit CA-Clipper-Code
                       # in einem Overlay  ablegen
   LIB extend             # CA-Clipper EXTEND-Bibliothek ins Overlay
ENDAREA                   # Ende des Overlay-Bereichs
```

Der wichtigste dieser Befehle ist BLINKER INCREMENTAL OFF. Das inkrementelle Linken ist für den Blinker standardmäßig aktiviert, um die Entwicklungsarbeit zu beschleunigen.

Während der Entwicklung einer Applikation ist diese Einstellung optimal.

Vor der Produktion der Endversion muß das inkrementelle Linken unbedingt deaktiviert werden. Inkrementelles Linken reduziert den verfügbaren Speicher ebenso wie das Linken mit Debugger-Informationen sowie die Ausführungsgeschwindigkeit einer Anwendung und ist nur für die Entwicklungsphase gedacht.

Die Option sollte nicht mehr eingesetzt werden, wenn mit Ihrer Anwendung Speicherplatzprobleme auftreten.

DOS-Extender-Programm / Programm ohne Overlays

```
# TEST.LNK (Das ist ein Kommentar)
# Für DOS-Real-Mode-Programm: Nächste Zeile auskommentieren
BLINKER EXECUTABLE EXTENDED   # Erzeugt ein DOS-Extender-
                           # Programm
BLINKER INCREMENTAL OFF    # Für die Endversion
FILE test
FILE limerick
SEARCH BLXCLPnn            # Blinker-Dual Mode-Bibliothek
```

Dadurch linkt Blinker die beiden über die FILE-Befehle angegebenen Objektmodule (test.obj und limerick.obj) zu einem DOS-Extender-Programm mit dem Namen TEST.EXE zusammen. TEST ist der Name der ersten .OBJ-Datei in einem FILE-Befehl.

Dual Mode-Programm / Overlay-Programm

```
# TESTDUAL.LNK
# Nur für dynamische Overlays: nächste Zeile auskommentiereen
BLINKER EXECUTABLE DUAL    # Erzeugt Dual Mode-Programm
```

```
BLINKER OVERLAY OPSIZE 50     # 50 KB Overlay-Bereich
BLINKER OVERLAY PAGEFRAME ON  # Overlays laufen im Pageframe
BLINKER MESSAGE WINK          # Nur mit einem Auge blinken
BLINKER INCREMENTAL OFF    # Für die Endversion
FILE test                  # Diese Datei nicht im Overlay
BEGINAREA                  # Beginn des Overlay-Bereichs
   FILE limerick           # Diese Datei im Overlay
ENDAREA                       # Ende des Overlay-Bereichs
SEARCH BLXCLPnn            # Blinker-Dual Mode-Bibliothek
OUTPUT t                   # Name der Ausgabe.EXE
MAP A,S                       # Erzeugt eine MAP -Datei
```

16-Bit Windows-Programm

```
# TESTWIN.LNK
BLINKER INCREMENTAL OFF# Für die Endversion
FILE test
FILE limerick

SEARCH clip4win
oder
@FIVEWIN

LIB <übrige Bibliotheken>
DEFFILE TEST            # Benutzt TEST.DEF


;TEST.DEF
NAME          TEST
DESCRIPTION      'Ein kleines Windows-Programm'
;   Die Anforderungen der .DEF-Datei finden
;   Sie in der Clip4Win- oder FiveWin-Dokumentation.
```

In Kapitel 8 dieses Handbuches finden Sie eine tiefergehende Erläuterung zur Verwendung von Blinker zur Erzeugung von CA-Clipper-Programmen.

Schnelleinstieg für verschiedene Compiler

FORCE

Blinker unterstützt FORCE-Versionen von 2.1 an aufwärts zur Erzeugung von DOS-Real Mode-Programmen. Die folgenden, in FORCE geschriebenen Beispielprogramme, werden während dieses Abschnittes verwendet, um Ihnen die Grundlagen des Linkens von FORCE-Programmen mit Blinker zu veranschaulichen.

TEST.PRG:

```
PROCEDURE limerick PROTOTYPE
PROCEDURE force_main
DO limerick
QUIT
ENDPRO
```

LIMERICK.PRG:

```
PROCEDURE limerick
? "Es war ein junger Mitarbeiter von Dvorak"
RETURN
ENDPRO
```

Diese beiden Module werden wie folgt kompiliert:

```
FORCE test
FORCE limerick
```

Blinker von der Kommandozeile aus starten:

```
BLINKER FILE test,limerick LIB force OUTPUT t
```

Das veranlaßt Blinker, die beiden Objektmodule, die nach dem FILE-Befehl angegeben wurden (test.obj und limerick.obj), zu linken. Das Ergebnis ist ein ausführbares Programm mit dem Namen t.exe.

Programm ohne Overlays

Die folgende Skriptdatei liefert das gleiche Ergebnis wie die eben geschilderte Ausführung des Blinkers von der Kommandozeile aus:

```
# TEST.LNK  (dies ist ein Kommentar)
FILE test
FILE limerick
LIB force           # FORCE Compiler-Bibliothek
OUTPUT t            # Name der Ausgabe.EXE
```

Um Blinker mit dieser Skriptdatei zu starten, geben Sie bitte folgenden Befehl ein:

```
BLINKER @test
```

Am '@'-Zeichen erkennt Blinker, daß nun der Name einer Skriptdatei folgt. Blinker nimmt als Voreinstellung die Erweiterung .LNK an, es sei denn, eine andere wurde in der Kommandozeile angegeben.

Programm mit Overlays

Overlay-Programm mit zusätzlichen Befehlen

```
# TESTOVL.LNK
# 50 KB Overlay-Bereich
BLINKER OVERLAY OPSIZE 50
# Overlays im Pageframe starten
BLINKER OVERLAY PAGEFRAME ON
BLINKER MESSAGE WINK          # Nur mit einem Auge blinken
BEGINAREA                     # Beginn des Overlay-Bereichs
        FILE test             # Diese Datei im Overlay
        FILE limerick         # Diese Datei im Overlay
ENDAREA                       # Ende des Overlay-Bereichs
LIB force                     # FORCE-Bibliothek wird nicht im
                              # Overlay abgelegt
OUTPUT t                      # Name der Ausgabe.EXE
STACK 4096                    # Stackgröße 4 KB
MAP A,S                       # Erzeugt eine MAP-Datei
```

Schnelleinstieg für verschiedene Compiler

Microsoft Assembler

Blinker unterstützt die Microsoft Assembler-Versionen von 4.0 an aufwärts. Anhand der folgenden in Microsoft Assembler geschriebenen Beispielprogramme werden in diesem Abschnitt die Grundlagen des Linkens von Microsoft Assemblerprogrammen mit Blinker veranschaulicht.

DOS-Extender-Programme, Dual Mode-Programme und Programme, die ins Overlay gelegt werden, müssen Far-Aufrufe nutzen, d.h. mit dem Medium, Large oder Huge Speichermodell assembliert werden. DOS-Extender und Dual Mode unterstützen das Medium Modell nicht.

TEST.ASM

```
.model large
. stack 100h
; Prozeduren für das Overlay müssen als 'Far' deklariert werden
extrn     limerick:far
.code
main      proc      far
          mov       ax,DGROUP
          mov       ds,ax
          call      limerick
          mov       ax,4c00h
          int       21h
main      endp
end       main
```

Beachten Sie bitte, daß es sich bei dem Startmodul um ein Assembler-Programm handelt, das nicht overlayfähig ist.

LIMERICK.ASM

```
.model        large
public        limerick
.data
testmsg       db        'Es war ein junger Mitarbeiter von Microsoft'
          db        0dh,0ah,'$'
.code
; Overlay-Prozeduren müssen vom Typ far sein
limerick  proc      far
          mov       ah,9
          mov       dx,offset DGROUP:testmsg
          int       21h
          ret
limerick  endp
end
```

Die beiden Module werden folgendermaßen assembliert:

```
MASM test;
MASM limerick;
```

DOS-Extender-Programm / Programm ohne Overlays

```
# TEST.LNK (Das ist ein Kommentar)
# Für DOS-Real-Mode-Programm: Nächste Zeile auskommentieren
BLINKER EXECUTABLE EXTENDED   # Erzeugt ein DOS-Extender-
                              # Programm
```

```
FILE test
FILE limerick
```

Das veranlaßt Blinker, die beiden Objektmodule, die nach dem FILE-Befehl angegeben wurden (test.obj und limerick.obj), zu linken. Das Ergebnis ist ein ausführbares Programm mit dem Namen test.exe, dem Namen der ersten im FILE-Befehl aufgeführten .OBJ-Datei.

Dual Mode-Programm / Programm mit Overlays

Beachten Sie bitte, daß das Startmodul eines Assembler-Programms nicht overlayfähig ist.

```
# TESTDUAL.LNK

# Nur für dynamische Overlays: nächste Zeile auskommentieren:
BLINKER EXECUTABLE DUAL# Erzeugt ein Dual Mode-Programm
BLINKER OVERLAY OPSIZE 50  # 50 KB Overlay-Bereich
BLINKER OVERLAY PAGEFRAME ON  # Overlays laufen im Pageframe
BLINKER MESSAGE WINK       # Nur mit einem Auge blinken
FILE test               # Diese Datei nicht im Overlay
BEGINAREA               # Beginn des Overlay-Bereichs
   FILE limerick        # Diese Datei im Overlay
ENDAREA                    # Ende des Overlay-Bereichs
OUTPUT t                # Name der Ausgabe.EXE
MAP A,S                    # Erzeugt eine MAP-Datei
```

16-Bit Windows-Programm

```
#TESTWIN.LNK
FILE test
FILE limerick
DEFFILE TEST     # Verwendet TEST.DEF

#TEST.DEF
NAME          TEST
DESCRIPTION      'Ein kleines Windows-Programm'
EXETYPE          WINDOWS 3.1
HEAPSIZE      1024
STACKSIZE     8192
```

Schnelleinstieg für verschiedene Compiler

Microsoft BASIC PDS 7.1 / VB DOS 1.0

Blinker unterstützt die Microsoft BASIC PDS-Version 7.1 sowie Visual Basic für DOS Version 1.0 zur Erzeugung von DOS-Real Mode-Programmen. Die folgenden in Microsoft BASIC PDS 7.1 geschriebenen Beispielprogramme werden während dieses Abschnitts benutzt, um Ihnen die Grundlagen des Linkes von Microsoft BASIC-Programmen mit Blinker zu veranschaulichen.

Wenn BASIC PDS-Programme mit Blinker erzeugt werden, muß die .OBJ-Datei, die die Hauptprozedur beinhaltet, immer im Root abgelegt werden. In einem Overlay-Programm müssen alle BASIC-Module mit dem Schalter -O kompiliert werden. Dadurch wird die Verwendung von CHAIN mit COMMON ausgeschlossen.

TEST.BAS:

```
DECLARE SUB limerick
```

```
CALL limerick

END
```

LIMERICK.BAS:

```
SUB limerick
print "Es war ein junger Mitarbeiter von Microsoft"
END SUB
```

Die beiden Module werden folgendermaßen kompiliert:

```
BC -o test;
BC -o limerick;
```

Blinker von der Kommandozeile aus starten:

```
BLINKER FILE test,limerick OUTPUT t
```

Das veranlaßt Blinker, die beiden Objektmodule, die im FILE-Befehl angegeben wurden (test.obj und limerick.obj), zu linken. Das Ergebnis ist ein ausführbares Programm mit dem Namen t.exe.

Programm ohne Overlays

Die folgende Skriptdatei liefert das gleiche Ergebnis wie die eben geschilderte Ausführung des Blinkers von der Kommandozeile:

```
# TEST.LNK  (Dies ist ein Kommentar)
FILE test
FILE limerick
OUTPUT t
```

Um Blinker zu veranlassen, diese Skriptdatei auszuführen, geben Sie bitte folgenden Befehl ein:

```
BLINKER @test
```

Am '@'-Zeichen erkennt Blinker, daß nun der Name einer Skriptdatei folgt. Blinker nimmt als Voreinstellung die Erweiterung .LNK an, es sei denn, in der Kommandozeile wurde eine andere Dateierweiterung angegeben.

Overlay-Programm mit zusätzlichen Befehlen

```
# TESTCMD.LNK
BLINKER OVERLAY OPSIZE 50 # 50 KB Overlay-Bereich
BLINKER OVERLAY PAGEFRAME ON # Overlays laufen im Pageframe
BLINKER MESSAGE WINK    # Nur mit einem Auge blinken
FILE test           # Hauptroutine muß ins Root
BEGINAREA           # Beginn des Overlay-Bereichs
   FILE limerick    # Diese Datei im Overlay
ENDAREA                # Ende des Overlay-Bereichs
OUTPUT t            # Name der Ausgabe.EXE
MAP A,S                # Erzeugt eine MAP-Datei
```

Schnelleinstieg für verschiedene Compiler

Microsoft QuickBASIC 4.5

Blinker unterstützt die Microsoft QuickBASIC-Version 4.5 zur Erzeugung von DOS-Real Mode-Programmen. Die folgenden, in Microsoft QuickBASIC 4.5 geschriebenen Beispielprogramme, werden während dieses Abschnittes verwendet, um Ihnen die Grundlagen des Linkens von Microsoft QuickBASIC-Programmen mit Blinker zu veranschaulichen.

Beachten Sie bitte, daß Blinker infolge der Einschränkungen in der QuickBasic-Laufzeitbibliothek kein dynamisches Overlaying von QuickBASIC in Verbindung mit der Fehlerbehandlung unterstützt, d.h., den Gebrauch von ON <Ereignis>- oder ON ERROR-Befehlen. Alle anderen Funktionen des Blinker werden jedoch von QuickBASIC voll unterstützt.

TEST.BAS:

```
DECLARE SUB limerick
CALL limerick
END
```

LIMERICK.BAS:

```
SUB limerick
print "Es war ein junger Mitarbeiter von Microsoft"
END SUB
```

Die beiden Module werden folgendermaßen kompiliert:

```
BC -o test;
BC -o limerick;
```

Start des Blinkers von der Kommandozeile aus:

```
BLINKER FILE test,limerick OUTPUT t
```

Das veranlaßt Blinker, die beiden Objektmodule, die im FILE-Befehl angegeben wurden (test.obj und limerick.obj), zu linken. Das Ergebnis ist ein ausführbares Programm mit dem Namen t.exe.

Programm ohne Overlays

Die folgende Skriptdatei liefert das gleiche Ergebnis, wie die eben geschilderte Ausführung des Blinkers von der Kommandozeile aus:

```
# TEST.LNK  (Dies ist ein Kommentar)
FILE test
FILE limerick
OUTPUT t
```

Um Blinker mit dieser Skriptdatei zu starten, geben Sie bitte folgenden Befehl ein:

```
BLINKER @test
```

Am '@'-Zeichen erkennt Blinker, daß nun der Name einer Skriptdatei folgt. Blinker nimmt als Voreinstellung die Erweiterung .LNK an, es sei denn, in der Kommandozeile wurde eine andere angegeben.

Programm ohne Overlays und mit zusätzlichen Befehlen

```
# TESTCMD.LNK
```

```
# Einbrennen einer Seriennummer
BLINKER EXECUTABLE SERIAL 123
BLINKER MESSAGE WINK       # nur mit einem Auge blinken
FILE test               # Hauptroutine
FILE limerick           # zweite Datei
OUTPUT t                # Name der .EXE-Datei
MAP A, S                # Erzeugen einer MAP-Datei
STACK 4096                 # Stack-Größe 4KB
```

Schnelleinstieg für verschiedene Compiler

Microsoft C / C++ / Quick C

BLINKER unterstützt die Microsoft C-Versionen von 5.0 an aufwärts und alle Versionen von Microsoft Quick C für DOS Realmode- und Windowsprogramme, sowie ab Version 5.1 von Microsoft C/C++ auch DOS Protected Mode Programme. Die folgenden, in Microsoft C geschriebenen Beispielprogramme dienen dazu, die Grundlagen des Linkens von Microsoft C- und Microsoft Quick C-Programmen mit Blinker zu veranschaulichen.

DOS-Extender-Programme, Dual Mode-Programme und Programme, die ins Overlay gelegt werden, müssen Far-Aufrufe nutzen, d.h. mit dem Medium, Large oder Huge Speichermodell kompiliert werden. DOS-Extender und Dual Mode unterstützen das Medium Modell nicht.

Die Microsoft C/C++ Grafik-Bibliotheken für Version 5.1 und später sind voll Protected-Mode fähig.

Für DOS-Extender-Programme, die in Quick C geschrieben wurden, sollten Sie die Bibliothek BLXMSC51 als BLXMSCnn Bibliothek verwenden.

TEST.C:

```
void limerick (void);
void main ()
{
   limerick ();
}
```

LIMERICK.C:

```
#include <stdio.h>
void limerick (void)
{
   printf ("Es war ein junger Mitarbeiter von Microsoft\n");
}
```

Beide Module werden im Large Modell-Code kompiliert:

Microsoft C

```
CL -c -AL test.c
CL -c -AL limerick.c
```

Quick C

```
QCL -c -AL test.c
QCL -c -AL limerick.c
```

DOS-Extender-Programm / Programm ohne Overlays

```
# TEST.LNK (Das ist ein Kommentar)
# DOS-Real-Mode-Programm: Nächste Zeile auskommentieren
BLINKER EXECUTABLE EXTENDED   # Erzeugt ein DOS-Extender-
                              # Programm
FILE test
FILE limerick
SEARCH BLXMSCnn               # Blinker-Dual Mode-Bibliothek
MIXCASE                          # Groß-/Kleinschreibung unterscheiden
```

Das veranlaßt Blinker, die beiden Objektmodule zu linken, die nach dem FILE-Befehl angegeben wurden (test.obj und limerick.obj). Das Ergebnis ist ein ausführbares Programm mit dem Namen TEST.EXE.

Dual Mode-Programm / Overlay-Programm

```
# TESTDUAL.LNK
# Nur für dynamische Overlays auskommentieren:
BLINKER EXECUTABLE DUAL# Erzeugt ein Dual Mode-Programm
BLINKER OVERLAY OPSIZE 50  # 50 KB Overlay-Bereich
BLINKER OVERLAY PAGEFRAME ON  # Overlays laufen im Pageframe
BLINKER MESSAGE WINK       # Nur mit einem Auge blinken
FILE test                # Diese Datei nicht im Overlay
BEGINAREA                # Beginn des Overlay-Bereichs
   FILE limerick         # Diese Datei im Overlay
ENDAREA                     # Ende des Overlay-Bereichs
SEARCH BLXMSCnn          # Blinker Dual Mode-Biblithek
OUTPUT t                 # Name der Ausgabe.EXE
MAP A,S                     # Erzeugt eine MAP-Datei
MIXCASE                     # Groß-/Kleinschreibung unterscheiden
```

16-Bit Windows-Programm

```
#TESTWIN.LNK

FILE test
FILE limerick
LIB llibcew, libw ...   # Windows-Bibliotheken
NODEFLIB             # keine Default-Bibliothek
MIXCASE                 # Groß-/Kleinschreibung unterscheiden
DEFFILE TEST         # Verwendet TEST.DEF

#TEST.DEF
NAME          TEST
DESCRIPTION      'Ein kleines Windows-Programm'
EXETYPE          WINDOWS 3.1
HEAPSIZE      1024
STACKSIZE     8192
```

Für Windows-Programme kann es in manchen Fällen erforderlich sein, daß Microsoft-Bibliotheken mit dem NODEFLIB-Befehl als nicht voreingestellte Bibliotheken in der Linkdatei angegeben werden müssen.

Schnelleinstieg für verschiedene Compiler

Microsoft FORTRAN

BLINKER unterstützt Microsoft FORTRAN von der Version 5.1 an aufwärts. Die folgenden, in Microsoft FORTRAN geschriebenen Beispielprogramme veranschaulichen in diesem Abschnitt die Grundlagen des Linkens von Microsoft FORTRAN-Programmen mit Blinker.

DOS-Extender-Programme, Dual Mode-Programme und Programme, die ins Overlay gelegt werden, müssen Far-Aufrufe nutzen, d.h. mit dem Medium, Large oder Huge Speichermodell übersetzt werden. DOS-Extender und Dual Mode unterstützen das Medium Modell nicht.

Die Microsoft FORTRAN Grafikbibliothekenfür Version 5.1 und später sind voll Protected Mode fähig.

TEST.FOR:

```
      CALL limerick
      END
```

LIMERICK.FOR:

```
      SUBROUTINE limerick ()
      WRITE (*,1000)
      WRITE (*,1010)
      RETURN
1000  FORMAT("Es war ein junger Mitarbeiter von")
1010  FORMAT(" Microsoft")
      END
```

Die beiden Module werden folgendermaßen als Large Modell-Code kompiliert:

```
FL /c /AL test.for
FL /c /AL limerick.for
```

DOS-Extender-Programm / Programm ohne Overlays

```
# TEST.LNK (Das ist ein Kommentar)
# DOS-Real-Mode-Programm: Nächste Zeile auskommentieren
BLINKER EXECUTABLE EXTENDED   # Erzeugt ein DOS-Extender-
                              # Programm
FILE test
FILE limerick
SEARCH BLXMSFnn               # Blinker Extended-Mode Bibliothek
```

Das veranlaßt Blinker, die beiden Objektmodule zu linken, die nach dem FILE-Befehl angegeben wurden (test.obj und limerick.obj). Das Ergebnis ist ein ausführbares Programm mit dem Namen TEST.EXE.

Overlay-Programm

```
# TESTOVL.LNK
BLINKER OVERLAY OPSIZE 50  # 50 KB Overlay-Bereich
BLINKER OVERLAY PAGEFRAME ON  # Overlays laufen im Pageframe
BLINKER MESSAGE WINK       # Nur mit einem Auge blinken
FILE test                # Diese Datei nicht im Overlay
BEGINAREA                # Beginn des Overlay-bereichs
   FILE limerick         # Diese Datei im Overlay
ENDAREA                     # Ende des Overlay-Bereichs
OUTPUT t                 # Name der Ausgabe.EXE
MAP A,S                     # Erzeugt eine MAP-Datei
```

Windows-Programm

```
#TESTWIN.LNK
FILE test
FILE limerick
LIB llibfew, libw ...   # Windows-Bibliotheken
NODEFLIB            # keine Default-Bibliothek
DEFFILE TEST        # Verwendet TEST.DEF
```

```
#TEST.DEF
NAME          TEST
DESCRIPTION      'Ein kleines Windows-Programm'
EXETYPE          WINDOWS 3.1
HEAPSIZE      1024
STACKSIZE     8192
```

Schnelleinstieg für verschiedene Compiler

Microsoft PASCAL

Blinker unterstützt die Microsoft PASCAL-Versionen von 4.0 an aufwärts. Die folgenden, in Microsoft PASCAL geschriebenen Beispielprogramme werden während dieses Abschnittes verwendet, um Ihnen die Grundlagen des Linkens von Microsoft PASCAL-Programmen mit dem Blinker zu veranschaulichen.

DOS-Extender-Programme, Dual Mode-Programme und Programme, die ins Overlay gelegt werden, müssen Far-Aufrufe nutzen, d.h. mit dem Medium, Large oder Huge Speichermodell assembliert werden. DOS-Extender und Dual Mode unterstützen das Medium Modell nicht.

TEST.PAS:

```
program test (input,output);
procedure limerick; extern;
begin
   limerick;
end.
```

LIMERICK.PAS:

```
module limtest;
procedure limerick;
begin
   writeln ('Es war ein junger Mitarbeiter von Microsoft');
end;
end.
```

Die beiden Module werden folgendermaßen kompiliert:

```
PL -c test.pas
PL -c limerick.pas
```

DOS-Extender-Programm / Programm ohne Overlays

```
# TEST.LNK (Das ist ein Kommentar)
# DOS-Real-Mode-Programm: Nächste Zeile auskommentieren
BLINKER EXECUTABLE EXTENDED   # Erzeugt ein DOS-Extender-Programm
FILE test
FILE limerick
SEARCH BLXMSPnn           # Blinker-Dual Mode-Bibliothek
```

Das veranlaßt Blinker, die beiden Objektmodule zu linken, die nach dem FILE-Befehl angegeben wurden (test.obj und limerick.obj). Das Ergebnis ist ein ausführbares Programm mit dem Namen TEST.EXE.

Dual Mode- / Overlay-Programm

```
# TESTDUAL.LNK
# Nur für dynamische Overlays auskommentieren:
BLINKER EXECUTABLE DUAL# Erzeugt ein Dual Mode-Programm
BLINKER OVERLAY OPSIZE 50  # 50 KB Overlay-Bereich
BLINKER OVERLAY PAGEFRAME ON  # Overlays laufen im Pageframe
BLINKER MESSAGE WINK       # Nur mit einem Auge blinken
FILE test               # Diese Datei nicht im Overlay
BEGINAREA               # Beginn des Overlay-Bereichs
   FILE limerick        # Diese Datei im Overlay
ENDAREA                    # Ende des Overlay-Bereichs
SEARCH BLXMSPnn         # Blinker-Dual Mode-Bibliothek
OUTPUT t                # Name der Ausgabe.EXE
MAP A,S                    # Erzeugt eine MAP-Datei
```

16-Bit Windows-Programm

```
#TESTWIN.LNK
FILE test
FILE limerick
LIB libpasew, libw ... # Windows-Bibliotheken
NODEFLIB            # keine Default-Bibliotheken
DEFFILE TEST        # Verwendet TEST.DEF

#TEST.DEF
NAME          TEST
DESCRIPTION      'Ein kleines Windows-Programm'
EXETYPE          WINDOWS 3.1
HEAPSIZE      1024
STACKSIZE     8192
```

Schnelleinstieg für verschiedene Compiler

Microsoft Visual C++

Blinker unterstützt die MicrosoftVisual C++-Versionen 1.x für Real- und Protectedmode DOS und Windows Programme. Die 32-Bit Versionen 2.x und später werden zur Zeit nicht unterstützt wegen des COFF-Formats der Objektdateien.

Die folgenden, in Microsoft Visual C++ geschriebenen Beispielprogramme werden während dieses Abschnittes verwendet, um Ihnen die Grundlagen des Linkens von Microsoft Visual C++-Programmen mit dem Blinker zu veranschaulichen.

DOS-Extender-Programme, Dual Mode-Programme und Programme, die ins Overlay gelegt werden, müssen Far-Aufrufe nutzen, d.h. mit dem Medium, Large oder Huge Speichermodell assembliert werden. DOS-Extender und Dual Mode unterstützen das Medium Modell nicht.

TEST.C

```
void limerick (void);
void main ()
```

```
{
   limerick ();
}
```

LIMERICK.C

```
#include <stdio.h>
void limerick (void)
{
   printf ("Es war ein junger Mann von Microsoft\n");
}
```

Diese beiden Module werden wie folgt im Large Model kompiliert :

Microsoft Visual C/C++

```
CL -c -AL test.c
CL -c -AL limerick.c
```

DOS Extender-Programm / Programm ohne Overlays

```
# TEST.LNK (Das ist ein Kommentar)
# DOS-Real-Mode-Programm: Nächste Zeile auskommentieren
BLINKER EXECUTABLE EXTENDED   # Erzeugt ein DOS-Extender-
                              # Programm
FILE test
FILE limerick
SEARCH BLXMVCnn               # Blinker-Dual Mode-Bibliothek
MIXCASE                          # Groß-/Kleinschreibung unterscheiden
```

Das veranlaßt Blinker, die beiden Objektmodule zu linken, die nach dem FILE-Befehl angegeben wurden (test.obj und limerick.obj). Das Ergebnis ist ein ausführbares Programm mit dem Namen TEST.EXE.

Dual Mode-Programm / Overlay-Programm

```
# TESTDUAL.LNK
# Nur für dynamische Overlays: nächste Zeile auskommentieren:
BLINKER EXECUTABLE DUAL# Erzeugt ein Dual Mode-Programm
BLINKER OVERLAY OPSIZE 50  # 50 KB Overlay-Bereich
BLINKER OVERLAY PAGEFRAME ON  # Overlays laufen im Pageframe
BLINKER MESSAGE WINK       # Nur mit einem Auge blinken
FILE test                 # Diese Datei nicht im Overlay
BEGINAREA                 # Beginn des Overlay-Bereichs
   FILE limerick          # Diese Datei im Overlay
ENDAREA                      # Ende des Overlay-Bereichs
SEARCH BLXMVCnn           # Blinker Dual Mode-Biblithek
OUTPUT t                  # Name der Ausgabe.EXE
MAP A,S                      # Erzeugt eine MAP-Datei
MIXCASE                      # Groß-/Kleinschreibung unterscheiden
```

16-Bit Windows-Programm

```
#TESTWIN.LNK

FILE test
FILE limerick
LIB llibcew, libw ...   # Windows-Bibliotheken
NODEFLIB             # keine Default-Bibliothek
MIXCASE                 # Groß-/Kleinschreibung unterscheiden
DEFFILE TEST         # Verwendet TEST.DEF

#TEST.DEF
NAME          TEST
```

```
DESCRIPTION      'Ein kleines Windows-Programm'
EXETYPE          WINDOWS 3.1
HEAPSIZE     1024
STACKSIZE    8192
```

Für Windows-Programme kann es in manchen Fällen erforderlich sein, daß Microsoft-Bibliotheken mit dem NODEFLIB-Befehl als nicht voreingestellte Bibliotheken in der Linkdatei angegeben werden müssen.

Schnelleinstieg für verschiedene Compiler

Stony Brook Modula 2

Blinker unterstützt alle Versionen des Stony Brook Modula für DOS Real Mode Programme.

Wenn Sie Overlays verwenden, müssen Sie im LARGE Model kompilieren und sicherstellen, daß das Startmodul im Root bleibt und nicht ins Overlay gelinkt wird :

```
m2 myprog.mod /CODE:L
```

Schnelleinstieg für verschiedene Compiler

Stony Brook Pascal+

Blinker unterstützt alle Versionen von Stony Brook Pascal+ für DOS Real Mode Programme.

Wenn Sie Overlays verwenden, müssen Sie im LARGE Model kompilieren und Konstanten in ein separates Segment plazieren:

```
ppc myprog.pas //CODE:L //CONSTANT:L
```

Schnelleinstieg für verschiedene Compiler

Symantec C++

Blinker unterstützt die Symantec C++-Versionen von 4.0 an aufwärts für DOS Real Mode Programme und Windowsprogramme, sowie Version 6.1 und später für Protected Mode Programme.

Die folgenden, in Symantec C++ geschriebenen Beispielprogramme veranschaulichen Ihnen in diesem Abschnitt die Grundlagen des Linkens von Symantec C++-Programmen mit Blinker.

Programme, die ins Overlay gelegt werden, müssen Far-Aufrufe nutzen, d.h. mit dem Medium, Large oder Huge Speichermodell kompiliert werden.

Hinwei: Die Stackgröße eines Symantec C++ Programms muß durch die Variable _stack innerhalb eines

Programms festgelegt werden, nicht mit dem STACK Linkbefehl. (z.B. extern "C" unsigned _stack = 8192;). Bitte suchen Sie in der Dokumentation von Symantec nach weiteren Details.

TEST.C:

```
void limerick (void);
void main (void)
{
   limerick ();
}
```

LIMERICK.C:

```
#include <stdio.h>
void limerick (void)
{
   printf ("Es war ein junger Mitarbeiter von Symantec\n");
}
```

Die beiden Module werden folgendermaßen als Large-Model Code kompiliert:

```
SC -c -ml test.c
SC -c -ml limerick.c
```

DOS-Extender-Programm / Programm ohne Overlays

```
# TEST.LNK (Das ist ein Kommentar)
# DOS-Real-Mode-Programm: Nächste Zeile auskommentieren
BLINKER EXECUTABLE EXTENDED   # Erzeugt ein DOS-Extender-
                              # Programm
FILE test
FILE limerick
SEARCH BLXSCPnn               # Blinker-Dual Mode-Bibliothek
MIXCASE                          # Groß-/Kleinschreibung unterscheiden
```

Das veranlaßt Blinker, die beiden Objektmodule zu linken, die nach dem FILE-Befehl angegeben wurden (test.obj und limerick.obj). Das Ergebnis ist ein ausführbares Programm mit dem Namen TEST.EXE.

Dual Mode-Programm / Overlay-Programm

```
# TESTDUAL.LNK
# Nur für dynamische Overlays auskommentieren:
BLINKER EXECUTABLE DUAL# Erzeugt ein Dual Mode-Programm
BLINKER OVERLAY OPSIZE 50  # 50 KB Overlay-Bereich
BLINKER OVERLAY PAGEFRAME ON  # Overlays laufen im Pageframe
BLINKER MESSAGE WINK       # Nur mit einem Auge blinken
FILE test                # Diese Datei nicht im Overlay
BEGINAREA                # Beginn des Overlay-Bereichs
   FILE limerick         # Diese Datei im Overlay
ENDAREA                     # Ende des Overlay-Bereichs
SEARCH BLXSCPnn             # Blinker-Dual Mode-Bibliothek
OUTPUT t                 # Name der Ausgabe.EXE
MAP A,S                     # Erzeugt eine MAP-Datei
MIXCASE                     # Groß-/Kleinschreibung unterscheiden
```

16-Bit Windows-Programm

```
#TESTWIN.LNK
FILE test
FILE limerick
LIB swl, libw ...       # Windows-Bibliotheken - vor V.4.0: zwl
NODEFLIB             # keine Default Bibliotheken
DEFFILE TEST         # Verwendet TEST.DEF
```

```
#TEST.DEF
NAME          TEST
DESCRIPTION       'Ein kleines Windows-Programm'
EXETYPE           WINDOWS 3.1
HEAPSIZE      1024
STACKSIZE     8192
```

Schnelleinstieg für verschiedene Compiler

WATCOM C/C++

BLINKER unterstützt WATCOM C/C++ ab der Version 8.0 für DOS Real Mode und Windowsprogramme, sowie die 16-Bit Version 10.0 und später für Protected Mode.

Die folgenden, in WATCOM C geschriebenen Beispielprogramme veranschaulichen in diesem Abschnitt die Grundlagen des Linkens von WATCOM C-Programmen mit Blinker.

Der DOSSEG Befehl muß für WATCOM C/C++ immer im Linkskript angegeben werden. Der Laufzeitfehler 'Null assignment detected' deutet in der Regel darauf hin, daß dieser DOSSEG Befehl fehlt.

Programme, die ins Overlay gelegt werden, müssen Far-Aufrufe nutzen, d.h. mit dem Medium, Large oder Huge Speichermodell kompiliert werden.

Beachten Sie bitte, daß die Datei FP7WCCnn.OBJ explizit zusätzlich zu BLXWCCnn.LIB gelinkt werden muß, falls KEINE Floating Point Emulation in einem Protected Mode Programm benötigt wird, d.h. wenn mit den Optionen -fpc, -fpi87, -fp2 oder –fpr kompiliert wird.

Der WATCOM Resource-Compiler WRC erfordert die -zm Option, um eine Ausgabe im 16-Bit MS Format zu erzeugen. Diese Ausgabe kann dann von dem Blinker RC Linkskript Befehl verwendet werden. Doch für 32-Bit Windows Programme muß WRC für den zweiten Durchlauf des Resource-Compilers verwendet werden, da WATCOM kein Standardformat für die 32-Bit Resourcedatei verwendet.

TEST.C:

```
void limerick (void);
void main (void)
{
   limerick ();
}
```

LIMERICK.C:

```
#include <stdio.h>
void limerick (void)
{
   printf ("Es war ein junger Mitarbeiter von WATCOM\n");
}
```

Die beiden Module werden folgendermaßen als Large-Model Code kompiliert:

```
WCC -ml test.c
WCC -ml limerick.c
```

DOS-Extender-Programm / Programm ohne Overlays

```
# TEST.LNK (Das ist ein Kommentar)
```

```
# DOS-Real-Mode-Programm: Nächste Zeile auskommentieren
BLINKER EXECUTABLE EXTENDED   # Erzeugt ein DOS-Extender-Programm
DOSSEG                      # Für WATCOM C erforderlich
FILE test
FILE limerick
SEARCH BLXWCCnn             # Blinker-Dual Mode-Bibliothek
MIXCASE                        # Groß-/Kleinschreibung unterscheiden
```

Das veranlaßt Blinker, die beiden Objektmodule zu linken, die nach dem FILE-Befehl angegeben wurden (test.obj und limerick.obj). Das Ergebnis ist ein ausführbares Programm mit dem Namen TEST.EXE.

Dual Mode-Programm / Overlay-Programm

```
# TESTDUAL.LNK
# Nur für dynamische Overlays: nächste Zeile auskommentieren:
BLINKER EXECUTABLE DUAL# Erzeugt ein Dual Mode-Programm
DOSSEG                  # Für WATCOM C/C++ erforderlich
BLINKER OVERLAY OPSIZE 50  # 50 KB Overlay-Bereich
BLINKER OVERLAY PAGEFRAME ON  # Overlays laufen im Pageframe
BLINKER MESSAGE WINK       # Nur mit einem Auge blinken
FILE test               # Diese Datei nicht im Overlay
BEGINAREA               # Beginn des Overlay-Bereichs
   FILE limerick        # Diese Datei im Overlay
ENDAREA                    # Ende des Overlay-Bereichs
SEARCH BLXWCCnn         # Blinker-Dual Mode-Bibliothek
OUTPUT t                # Name der Ausgabe.EXE
MAP A,S                    # Erzeugt eine MAP-Datei
MIXCASE                    # Groß-/Kleinschreibung unterscheiden
```

16-Bit Windows-Programm

```
#TESTWIN.LNK
DOSSEG           # Für WATCOM C/C++ erforderlich
FILE test
FILE limerick
MIXCASE             # Groß-/Kleinschreibung unterscheiden
DEFFILE TEST     # Verwendet TEST.DEF

#TEST.DEF
NAME          TEST
DESCRIPTION      'Ein kleines Windows-Programm'
EXETYPE          WINDOWS 3.1
HEAPSIZE      1024
STACKSIZE     8192
```

Schnelleinstieg für verschiedene Compiler

Zinc Application Framework für DOS

Blinker unterstützt die Zinc Application Framework für DOS ab Version 5.12 für DOS Protected Mode Programme, wenn sie zusammen mit Borland C++ 5.x, Symantec C++ 7.x (von Usern unterstützt laut Zinc Dokumentation) oder WATCOM C++ 16-Bit ab Version 10.6 eingesetzt wird.

Falls es bei einem ZINC Programm, das mit dem Blinker Extender gelinkt ist, zu Fehlern kommt, versuchen Sie BITTE, dasselbe Programm mit Ihrem bisherigen Extender oder Linker, soweit vorhanden, zu linken, um zu sehen, ob das Problem weiterhin besteht.

Um Linkzeitfehler leichter isolieren zu können, verwenden Sie bitte die Option /K für ZMAKE, um die generierten temporären Linkskripte zu speichern.

Einstellungen

Stellen Sie zunächst sicher, daß die INCLUDE Umgebungsvariable auf das C Unterverzeichnis des Blinker-Verzeichnisses zeigt, oder kopieren Sie die Headerdateien (C*.h) in das Include-Verzeichnis Ihres Compilers. Überprüfen Sie ebenfalls, ob die PATH Umgebungsvariable auf das BIN Unterverzeichnis von Blinker zeigt.

Bevor Sie Blinker mit dem Zinc ZMAKE Make-Programm und den .MAK Skripten verwenden, müssen Sie noch die entsprechenden Blinker Protected Mode Dateien für den Compiler, den Sie benutzen, in das LIB Unterverzeichnis des Compilers kopieren. Die Namen der Blinker Protected Mode Dateien enden alle mit einer zweistelligen Versionsnummer des Compilers, die Sie zum Kopieren entfernen müssen.

Z.B. benötigt Blinker die Dateien C0LPnn.OBJ und BLXBCPnn.LIB aus seinen entsprechenden Unterverzeichnissen OBJ und LIB für Borland C/C++, wobei nn für die Versionsnummer des Compilers steht. Diese Dateien müssen in das Borland C/C++ LIB Unterverzeichnis kopiert werden ohne nn am Ende des Namens, z.B. für Version 5.x von Borland C/C++:

```
COPY C:\BLINKER\OBJ\C0LP50.OBJ C:\BORLANDC\LIB\C0LP.OBJ
COPY C:\BLINKER\LIB\BLXBCP50.LIB C:\BORLANDC\LIB\BLXBCP.LIB
```

Oder für WATCOM C/C++ 10.5

```
COPY C:\BLINKER\LIB\BLXWCC10.LIB C:\WATCOM\LIB16\BLXWCC.LIB
```

ZMAKE kann nun die Zinc Application Framework Bibliotheken und jedes der Beispielprogramme mit dem Ziel DOS16B erzeugen.

Erstellung eines Zinc Application Framework Beispielprogrammes

Beginnen Sie mit der Erstellung der Zinc Application Framework Bibliotheken aus dem SOURCE Unterverzeichnis des Zinc Application Framework Verzeichnisses.

```
CD \ZAF51\SOURCE
ZMAKE DOS16B
```

Dann wechseln Sie in das Verzeichnis, welches das Beispielprogramm enthält, und erstellen es mit demselben ZMAKE Befehl und Ziel, z.B.

```
CD \ZAF51\EXAMPLE\HELLO
ZMAKE DOS16B
```

Erstellung eigener Programme

Idealerweise kopieren Sie einfach die ZMAKE.MAK Datei aus einem der Zinc Application Framework Beispielprogramme, modifizieren sie für Ihr eigenes Programm und verwenden ZMAKE DOS16B zur Erstellung der Applikation.

Wenn Sie Ihre eigenen Make Dateien und Linkskripte für eine Zinc Application Framework Applikation schreiben wollen, verwenden Sie die gleichen Compileroptionen wie für den PharLap 286 Extender, so wie es von Zinc dokumentiert ist. Als Vorlage für Ihre Linkskripte können Sie die mit ZMAKE generierten und mit der Option /K (keep) gespeicherten Skripte irgend eines Zinc Application Framework Beispielprogrammes verwenden.

Stellen Sie sicher, daß Sie alle Befehlszeilenoptionen, die ZMAKE zur Blinker Befehlszeile hinzufügt, übernehmen.

ZMAKE erzeugt Linkskripte im MS Link, TLink oder WLink Format für alle Beispielprogramme. Um MS Linkskripte verarbeiten zu können, benötigt Blinker als ersten Parameter inder Befehlszeile den Parameter '/qm' oder die Einstellung 'LINKER=MSLINK' in der BLINKER.INI Datei. Um TLink Linkskripte verarbeiten zu können, benötigt Blinker als ersten Parameter inder Befehlszeile den Parameter '/qt' oder die Einstellung 'LINKER=TLINK' in der BLINKER.INI Datei. Um WLink Linkskripte verarbeiten zu können, benötigt Blinker als ersten Parameter inder Befehlszeile den Parameter '/qw'.

Alternativ können Sie auch das Blinker Standardformat, auch bekannt als freies Format, verwenden. Dieses Format ist in der Blinker Online-Hilfe und im Blinker Handbuch beschrieben. Daher suchen Sie bitte dort nach weiteren Einzelheiten.

Einfache Beispiele für DOS Extended Linkskripte für die unterschiedlichen unterstützten Compiler finden Sie in der Dokumentation oder der Online-Hilfe unter 'Compiler-Schnelleinstieg' im Kapitel 'Schnelleinstieg'. Die einzigen Befehle, die zusätztlich erforderlich sind, sind die LIB Befehle für die Zinc Application Framework Bibliotheken.

Ein Beispiel-Linkskript für das Zinc Application Framework Beispiel HELLO1 kompiliert mit Borland C++ 5.x befindet sich im ZAF Unterverzeichnis von Blinker :

```
#
#  HELLO.LNK     ZAF 5.1 HELLO Linkskript für BC++ 5.x
#
file c0lp50.obj  # Blinker Ersatzstartmodul für BC++ 5.x
file hello1.o16  # OBJ Dateinamen
out hello1          # Ausgabe Dateinamen
map s,a          # MAP-Datei erzeugen
bli exe ext      # DOS Extended Programm erzeugen
lib blxbcp50     # Blinker Protected Mode Bibliothek für BC++ 5.x
lib zafd16p      # ZAF Bibliotheken
lib zafd16
lib met_xp2d
lib met_pp2l
lib emu          # Borland Bibliotheken
lib mathl
lib cl
codeview         # Debugging Informationen einschließen
nodef            # Keine Default Bibliotheken
nobell           # Kein Piepsen
```

Für die anderen Compiler entfernen Sie einfach die Zeile 'file c0lp50.obj' und ersetzten Sie ‘blxbcp50’ in der Zeile 'lib blxbcp50' mit der entsprechenden Bibliothek für Ihren Compiler und Ihre Version, so wie es in der Blinker Dokumentation beschrieben ist.

Debuggen von Quelltext

Um eine Ausgabedatei mit Debugging-Informationen zu erhalten, setzten Sie DEBUG=ON in der ZMAKE Makedatei, oder verwenden Sie den CODEVIEW Befehl, wenn Sie ein Blinker Linkskript verwenden.

Um ein DOS Extended Programm zu debuggen, benötigen Sie einen Protected Mode fähigen Debugger, wie z.B. den NuMega SoftICE/W, weil die Standardebugger der Compiler zur Zeit nicht mit durch Blinker erzeugte DOS Extended Programme funktionieren.

Vollständige Debuginformationen einschließlich Zeilennumern, globale und lokale Variablen, Klassen und Strukturen sind in Symantec und WATCOM C/C++ Programmen verfügbar.

Borland C/C++ benutzt ein eigenes Format für die Debug-informationen von Symbolen. Daher unterstützt Blinker zur Zeit nur Zeilennummern uind globale Variablen unter Borland C/C++.

Unterstützung aller Debuginformationen unter Borland C/C++ ist für eine künftige Blinkerversion geplant.

Real Mode-Programme

Linken von Real Mode-Programmen

Lademodulgröße, Größe der ausführbaren Datei und Speicher

Code für Overlays auswählen

Linken von Real Mode-Programmen

Durch die Architektur der frühen IBM PC's und DOS ist die maximale Größe des konventionellen Speichers auf 640 KB begrenzt, obwohl manche Speichermanager sicher weitere 100 oder 200 KB auf einigen Rechnern zur Verfügung stellen könnten. Theoretisch ist der maximale konventionelle Speicher für den 8088/8086-Prozessor durch den 1 MB-Adreßraum beschränkt.

Häufig ist es der Fall, daß ein Computerprogramm viel zu groß wird, um noch in den verfügbaren Speicher hineinzupassen, oder daß es das Design erfordert, daß das Programm nur eine beschränkte Speichermenge nutzt, weil es zum Beispiel in einer Netzwerkumgebung läuft. Wenn es nicht möglich ist, ein Protected Mode-Programm zu erzeugen, um diese Speicherbegrenzungen zu überwinden, dann kann sich der Programmierer eine der leistungsfähigsten Features des Blinker zunutze machen - die dynamischen Overlays.

Durch Overlaying (Überlagerung) können sich gegenseitig ausschließende Programmteile während der Ausführung in einen gemeinsamen Speicherbereich geladen werden. Dies macht es einem Programm, das größer als der verfügbare Speicher ist, möglich, jeweils einen Abschnitt auszuführen. Unter Overlaying versteht man das Aufteilen des Codes in einzelne Abschnitte, wobei nur der Abschnitt in den Speicher geladen wird, der gerade benötigt wird. Der Hauptabschnitt verbleibt permanent im Speicher und die Overlay-Bereiche werden im Bedarfsfall in den EMS-, bzw. XMS-Bereich oder auf die Festplatte ausgelagert.

Linken von Real Mode-Programmen

Statische Overlays

Statisches Overlaying von Code stellt die traditionelle Methode des Overlaying dar, die erst vor kurzer Zeit durch das dynamische Overlaying abgelöst wurde. Ein statischer Overlay-Linker reserviert gewöhnlich einen Bereich im Root des ausführbaren Programmes. In diesem Bereich wird das Overlay geladen und gestartet. Die Größe des reservierten Bereiches innerhalb des Programmabbildes wird durch die vom Programmierer angelegte Overlay-Struktur und das Ausmaß des größten Overlay-Bereiches bestimmt. Wenn also ein Overlay größer ist als das andere, so kann dadurch eine bedeutende Menge des

Speichers verschwendet werden. Jedes Overlay besteht aus einer oder mehreren .OBJ-Dateien. Overlays können in Bereichen zusammengefaßt oder ineinander verschachtelt werden. Letzteres verkompliziert jedoch die Overlay-Struktur.

Linken von Real Mode-Programmen

Dynamische Overlays

Dynamische Overlays machen eine Strukturierung von Overlays überflüssig und beheben viele Probleme, die mit statischen Overlays auftraten. Der Entwickler muß zur Linkzeit nur angeben, was überlagert werden soll, den Rest erledigt der Overlay-Manager. Ein Speicherbereich wird für Overlays nur zur Laufzeit des Programms reserviert und bei jedem Programmlauf lädt der Overlay-Manager die Overlays in der Reihenfolge in diesen Bereich, in der sie aufgerufen wurden.

Der Overlay-Manager verwaltet eine Tabelle mit den Overlays und deren Position im Overlay-Bereich. Zusätzlich werden auch Informationen darüber gesammelt, ob ein Overlay derzeit benutzt wird bzw. wie oft und wann es zuletzt benutzt wurde. Wenn der Overlay-Bereich vollständig belegt ist, werden Overlays unter Berücksichtigung der Häufigkeit ihrer Nutzung und des Zeitpunktes der letzten Nutzung aus dem Speicher entfernt, um Platz für neue Overlays zu schaffen. Dieser Vorgang wird als dynamischer Overlay-Prozeß bezeichnet, da das Laden und Entfernen von Overlays aus dem Speicher bei jedem Neustart einer Anwendung variiert.

Der Overlay-Manager des Blinkers nimmt das Laden und Ausführen eines Overlays in vier Stufen vor:

1. Feststellen, ob das Overlay sich im Speicher befindet. Wenn das der Fall ist, weiter bei Punkt 4.

2. Suchen des Overlay-Codes im Overlay-Cache oder der Overlay-Datei; Laden des Codes an eine geeignete Stelle im Speicher innerhalb des Overlay-Pools.

3. Umsetzten aller im Code enthaltenen Verweise auf externe Variablen oder Prozeduren, d. h. solche außerhalb des Overlay-Codes. Dies erfolgt über eine Tabelle mit externen Referenzen, die innerhalb des Overlay-Codes gespeichert ist. Diese Methode entspricht der Art und Weise, wie DOS zur Laufzeit Referenzen auf bestimmte Segmente innerhalb einer .EXE-Datei erstellt, wenn die Datei geladen wird.

4. Übergabe der Steuerung an das neu geladene Overlay. Nach der Ausführung des Overlays verzweigt die Steuerung wieder an den Overlay-Manager zurück.

Die Verwendung eines separaten Overlay-Pools bringt doppelten Nutzen. Erstens kann sich eine größere Zahl von Overlays gleichzeitig ausführungsbereit im Speicher befinden, da dynamische Overlays nicht an eine bestimmte, zur Linkzeit festgelegte Adresse geladen werden müssen (im Gegensatz zu statischen Overlays), sondern stattdessen irgendwo in den Overlay-Pool geladen werden können. Das steigert die Ausführungsgeschwindigkeit. Zweitens kann die Größe des Overlay-Pools auf die verschiedenen Speicheranforderungen zur Laufzeit angepaßt werden.

Blinker erlaubt dem Entwickler, den Code einer Applikation unter optimaler Speicherausnutzung in dynamischen Overlays zu plazieren, ohne die Komplexität von Overlay-Strukturen in Kauf nehmen zu müssen. Das geht problemlos mit Hilfe eines BEGINAREA- und eines ENDAREA-Befehls. Alle Bibliotheken und Dateien, die zwischen diesen beiden Befehlen angegeben wurden, werden in einem Overlay abgelegt. Der gesamte Code der Bibliotheken und Dateien, der außerhalb dieser Overlay-Struktur plaziert wurde, verbleibt während der Ausführung permanent im Speicher.

```
FILE rootfile       # verbleibt im Speicher
```

```
BEGINAREA
   FILE File-x      # wird im Overlay abgelegt
   FILE Lib-y          # wird im Overlay abgelegt
ENDAREA
LIB Rootlib            # verbleibt im Speicher
```

Um eine Bibliothek im Overlay abzulegen, muß sie zwischen den BEGINAREA- und ENDAREA-Anweisungen plaziert werden. Dabei ist es egal welcher Befehl (LIBRARY oder SEARCH) benutzt wird, um die Bibliothek zu spezifizieren. Blinker besitzt die Fähigkeit, bei Bedarf Teile einer Bibliothek ins Overlay zu legen (Siehe Kapitel 9, der MODULE-Befehl).

Beim Linken mit dynamischen Overlays wird in der .EXE-Datei kein Platz für Overlays reserviert, wie es bei statischen Overlays der Fall ist. Blinker benutzt eine ‘feste’ Overlay-Belegungsmethode, wobei ein DOS-Speicherblock festgelegter Größe vom Overlay-Manager für den Overlay-Pool belegt wird, bevor das Hauptprogramm die Steuerung übernimmt. Dieser Speicherblock wird normalerweise an den Anfang des freien DOS-Speichers gelegt, es sei denn, der Programmierer gibt die UMBs oder den EMS-Pageframe für die Overlay-Belegung an. (Die einzige Ausnahme ist in Kapitel 8 beschrieben: CA-Clipper Sommer ’87 verwendet die dynamische Overlay-Belegungsmethode)

Für dynamische Overlays wird eine einzelne .OBJ-Datei als Einheit verwendet, d. h., Blinker lädt und führt Overlays auf einer Objektmodul, oder genauer, auf einer Codesegmentbasis und nicht auf Sektionsbasis aus, die gewöhnlich in einer statischen Overlay-Struktur verwandt wird. Blinker löst Kreuzverweise innerhalb von .OBJ-Dateien dynamisch auf, was zu einer größeren Speichereffizienz führt.

Hinweis für CA-Clipper:

Da jede Clipper-Prozedur oder -Funktion in einem separaten Segment enthalten ist, verwaltet Blinker überlagerten CA-Clipper-Code auf Prozedur- oder Funktionsebene, anstatt auf .OBJ-Dateiebene, wie es bei statischen Overlays der Fall ist. Dies erlaubt eine wesentlich effizientere Ausnutzung des Speichers und ein viel schnelleres Laden von angeforderten Overlays.

Linken von Real Mode-Programmen

Der Overlay-Pool

Der Speicherbereich, in dem die Overlays ausgeführt werden, wird als Overlay-Pool bezeichnet. Er kann im konventionellen Speicher, im UMB oberhalb von 640 KB oder in der EMS-Pageframe (mit einigen Einschränkungen hinsichtlich der EMS-Nutzung) angelegt werden. Die Größe des Overlay-Pools wird entweder zur Linkzeit über einen Befehl in der Skriptdatei angegeben oder zur Laufzeit durch die BLINKER-Umgebungsvariable.

Die minimale Größe des Overlay-Pools wird vom Blinker für das jeweilige Programm zur Linkzeit errechnet und angezeigt. Sie ist abhängig von der Größe des größten in einem Overlay abgelegten Objektmoduls. Wurde zur Linkzeit keine Größe angegeben, dann setzt Blinker die Größe auf die Voreinstellung von 40 KB oder auf den errechneten Minimalwert, je nachdem, welcher der größere ist. Wurde eine Größe für den Pool angegeben, dann wird die Größe auf den spezifizierten Wert oder auf den Minimalwert gesetzt, je nachdem, welcher von beiden größer ist.

Wenn Overlays nicht ausgeführt werden, werden sie entweder auf der Festplatte oder in einem Cache im EMS- oder XMS-Speicher abgelegt.

Linken von Real Mode-Programmen

Caching von Overlays

Wenn der Overlay-Cache aktiviert ist, beschleunigt er die Ausführung von Overlay-Programmen, indem er aus dem Overlay-Pool entfernte Overlays in den EMS- oder XMS-Speicher holt. Von hier aus können sie viel schneller neu geladen werden als von der Festplatte.

Während der Programmausführung muß der Overlay-Manager häufig ein Overlay aus dem Speicher entfernen, um Platz für ein anderes zu schaffen. Ohne Overlay-Cache entfernt Blinker die zuletzt am wenigsten angeforderten Overlays aus dem Speicher und lädt das neue Overlay in den frei gewordenen Speicher. Wenn der Overlay-Cache aktiviert ist, kopiert der Overlay-Manager sie in den Cache-Speicher. Ist der Cache voll, dann werden die zuletzt am wenigsten benötigten Overlays aus dem Cache entfernt, um für die neuen Platz zu schaffen.

Der Blinker Overlay-Cache benötigt entweder einen EMS-Manager (ab EMS 4.0 aufwärts) oder einen XMS-Manager (ab XMS 2.0 aufwärts). Zusätzlich kann einer der folgenden Befehle erforderlich sein:

```
BLINKER CACHE XMS x[%] [,y[%]]
```

und/oder

```
BLINKER CACHE EMS x[%] [,y[%]]
```

Der erste Parameter gibt die gewünschte Größe des angeforderten Cache entweder in KB oder als Prozent vom verfügbaren Arbeitsspeicher an. Der zweite Parameter beinhaltet den Teil des Hauptspeichers, der freiblieben soll. Der zweite Parameter überschreibt den ersten. Sollen beispielsweise 256 KB Cache im EMS belegt werden und und die Hälfte des EMS freigelassen werden, dann geben Sie an:

```
BLINKER CACHE EMS 256,50%
```

Stand dabei ein MB EMS zur Verfügung, dann würden 256 KB für den Cache alloziiert werden. Würden aber nur 368 KB zur Verfügung stehen, dann würden nur 184 KB (368 KB * 50%) belegt werden.

Es ist ratsam, in dem Link-Skript sowohl EMS als auch XMS anzugeben. Stehen dem Programm beide Speichertypen zur Verfügung, dann verwendet der Blinker standardmäßig XMS. Dadurch erreicht der Blinker die bestmögliche Verwendung der verfügbaren Speicherressourcen für das Caching. Wenn dem Blinker einer oder beide Speichertypen zur Programmlaufzeit nicht zur Verfügung stehen, dann ignoriert der Blinker die Cache-Anforderungen einfach.

Bitte beachten Sie: Das Overlay-Caching des Blinker ist einfach eine Methode zur Erhöhung der Arbeitsgeschwindigkeit einer Anwendung. Durch dieses Feature wird der Speicherbedarf der Anwendung nicht verringert. Mit einem genügend großen Cache-Speicher kann die optimale Overlay-Größe eines Programms jedoch auf ihr Minimum reduziert werden, wobei nur ein sehr kleiner Geschwindigkeitsverlust eintritt.

Optimale Cache-Größe

Idealerweise sollte der Cache groß genug sein, um den gesamten Overlay-Code des Programms aufnehmen zu können. Wenn das Programm lange genug läuft, befinden sich irgendwann einmal alle Overlays im Cache. Das Programm erreicht dadurch eine optimale Arbeitsgeschwindigkeit.

Die einfachste Möglichkeit, um dies zu erreichen, besteht darin, eine Cache-Größe anzugeben, die größer als die Gesamtgröße aller Overlays des Programms ist. Diese Größe wird über den MAP-Befehl unter der Überschrift "Overlays" ausgegeben. Der als "total" markierte Wert entspricht der Gesamtgröße aller Overlays, die über den Blinker gemanagt werden. Dieser Wert entspricht ebenfalls der maximal

erforderlichen Cache-Größe. Blinker belegt niemals mehr Cache-Speicher, als über die Gesamtgröße angegeben wurde, auch wenn er dazu aufgefordert wird.

Für die meisten Programme ist eine Cache-Größe von 256 KB ausreichend. Ein größerer Cache (bis zur Größe des Programms) erhöht die Vorteile.

Hinweis: Der Speicher für den Overlay-Cache des Blinker wird angefordert, bevor das Programm die Steuerung bekommt.

CA-Clipper 5.x

Wird für ein CA-Clipper 5.x-Programm Cache-Speicher angefordert, dann erhöht der Cache lediglich die Arbeitsgeschwindigkeit der Overlays, die vom Blinker-Overlay-Manager verwaltet werden. Wird ein CA-Clipper 5.x-Programm mit der Schalterstellung Inkrementelles Linken AUS gelinkt, dann wird sämtlicher CA-Clipper-Code über das Paging-System von CA-Clipper in Overlays abgelegt. Der Blinker verwaltet nur den C- und Assembler-Code des Programms.

Beim Einrichten eines Blinker-Cache muß darauf geachtet werden, daß genügend Platz für die Verwendung des VMM bleibt. Ansonsten wird der Cache durch den VMM beeinträchtigt, was sich ungünstig auf die Geschwindigkeit des Blinkers auswirkt. Ein gewöhnliches CA-Clipper-Program benötigt 500 KB bis 1 MB EMS für den VMM, um das Programm und dessen Daten effizient verwalten zu können.

Linken von Real Mode-Programmen

Interne und externe Overlays

Es gibt zwei Methoden für den Einbau von Overlays in Programme: die interne und die externe. Der Blinker erzeugt standardmäßig ein internes Overlay, indem die Overlays innerhalb der .EXE-Datei abgelegt werden und dadurch eine große .EXE-Datei erzeugt wird.

Alternativ können alle Overlays in einer oder mehreren separaten Overlay-Dateien abgelegt werden. Sie befinden sich außerhalb der .EXE-Datei und besitzen standardmäßig die Dateierweiterung .OVL. Diese externen Overlays werden durch den Einsatz eines oder mehrerer SECTION INTO-Befehle erzeugt, wobei jedes externe Overlay zur Laufzeit sein eigenes Datei-Handle benötigt. Diese Möglichkeit wird meistens genutzt, um den Vertrieb zu vereinfachen, da sie dem Entwickler ermöglicht, den ausführbaren Teil eines Programms auf verschiedenen Disketten auszuliefern.

Glücklicherweise muß der Blinker-Anwender die Organisation dieser Overlays nicht übernehmen: Das erledigt alles der Overlay-Manager zur Laufzeit. Der Overlay-Manager wird automatisch in das Programm gelinkt, wenn es Overlays enthält. Für die Erzeugung eines Overlay-Programms mit Hilfe des Blinkers, muß der Entwickler lediglich spezifizieren, welche Teile ins Overlay gelegt werden sollen und ob der Code in einem Objektmodul oder einer Bibliothek enthalten ist.

Lademodulgröße, Größe der ausführbaren Datei und Speicher

Für Programme mit dynamischen Overlays ergibt sich die Menge des benötigten Speichers aus drei Faktoren: (1) der Lademodulgröße, (2) dem Overlay-Pool und (3) zusätzlich vom Programm dynamisch belegtem Speicher. Beachten Sie, daß bei dynamischen Overlay-Linkern im Gegensatz zu statischen Overlay-Linkern die Größe des Overlay-Pools nicht in die Berechnung der Lademodulgröße einfließt.

Lademodulgröße, Größe der ausführbaren Datei und Speicher

Lademodulgröße

Der Wert, der zur Linkzeit in Klammern angezeigt wird, ist die "Lade"modulgröße der .EXE-Datei. Dies ist die minimale Menge konventionellen Speichers, die von DOS benötigt wird, um ein Abbild des Programms in den Speicher zu laden. Die vom Programm zur Laufzeit benötigte Speichermenge überschreitet häufig den zur Linkzeit angegebenen Wert.

Es ist schwer, die Lademodulgröße für eine zuverlässige Schätzung des tatsächlich zur Programmlaufzeit benötigten Speichers heranzuziehen, besonders bei den Compilern, die dynamisch ausgerichtet sind. Die Lademodulgröße ist eher als ein relativer Wert zu sehen, dessen Inhalt von Linklauf zu Linklauf variieren kann. Wenn sich der Wert erhöht, benötigt das Programm auch für gewöhnlich mehr Speicher, umgekehrt weniger. Da es eine allgemeine Konvention zur Festlegung der Lademodulgröße gibt, kann dieser Wert als Maßstab beim Vergleich unterschiedlicher Linker dienen.

Vom Code eines ins Overlay gelegten Objektmoduls wird nichts zu der Lademodulgröße hinzuaddiert. Jedoch werden alle Datensegmente innerhalb eines Overlay-Objektmoduls und alle in Hochsprachenmodulen definierten Daten im Root abgelegt. Somit vergrößert sich die Lademodulgröße, wenn dem Programm neue Daten hinzugefügt werden, ohne Rücksicht darauf, ob Daten aus einer Overlay-Prozedur stammen oder nicht. Ebenso kann, wenn ein weiteres Overlay-Objektmodul hinzukommt, die Einbeziehung irgendwelcher neu aufgerufener Funktionen, die sich in nicht overlayfähigen Bibliotheken befinden, zu einer Erhöhung der Lademodulgröße führen. Werden neue Objektmodule in den Overlay-Bereich eingefügt, hat dies einige neue Einträge in der internen Tabelle des Blinkers zur Folge, wodurch sich wiederum die Lademodulgröße leicht erhöht.

Eine ungewöhnlich große Veränderung der Lademodulgröße von einem Linklauf zum anderen sollte untersucht werden. Mögliche Ursachen sind dabei doppelte Funktionen in den .OBJ-Dateien, eine Änderung der Bibliotheksanordnung, oder, wenn Clipper im Einsatz ist, das Ein- und Ausschalten des inkrementellen Linkens.

Hinweis zu CA-Clipper:

Unmittelbar nach dem Laden belegt CA-Clipper zusätzliche Speicher, dessen Größe von Applikation zu Applikation variiert. Ungefähr 200 KB können für die Durchschnittsgröße von CA-Clipper Sommer ’87 Applikationen erwartet werden, einschließlich der Speicherbelegung durch die ‘R’- und ‘V’-Bereiche und den freien Pool. Das CA-Clipper 5.x VMM-System nutzt zusätzlich zum konventionellen Speicher EMS oder eine Auslagerungsdatei auf der Festplatte.

Lademodulgröße, Größe der ausführbaren Datei und Speicher

Der Overlay-Pool von Blinker

Die Größe des Overlay-Pools beträgt per Voreinstellung 40 KB. Sie kann aber zur Linkzeit über den Befehl BLINKER OVERLAY OPSIZE eingestellt werden oder auch zur Laufzeit durch Einsatz der BLINKER-Umgebungsvariablen, z.B.

```
SET BLINKER=/OO20
```

Zur Linkzeit zeigt Blinker den kleinstmöglichen Wert für die optimale Overlay-Größe an, der von der Größe des größten Overlay-Objektmoduls abhängt. Weitere Informationen über den Overlay-Pool finden Sie im gleichnamigen Abschnitt weiter oben in diesem Kapitel.

Hinweis zu CA-Clipper:

Für eine reine CA-Clipper 5.x-Applikation kann der Blinker Overlay-Pool vollständig eliminiert werden, indem lediglich der reine CA-Clipper-Code und keine Module aus EXTEND.LIB oder CLIPPER.LIB ins Overlay gelegt werden. In diesem Fall wird die minimale und augenblickliche Größe nicht zur Linkzeit angezeigt, da CA-Clipper 5.x das Overlaying von CA-Clipper-Code in Programmen, die nicht inkrementell gelinkt werden, selbst verwaltet.

Lademodulgröße, Größe der ausführbaren Datei und Speicher

Größe der .EXE-Datei

Die Größe der ausführbaren Datei ist bei Overlay-Programmen niemals ein Hinweis auf den tatsächlichen Speicherbedarf. Die .EXE-Datei kann zum Beispiel zusätzliche Informationen beinhalten, die jedoch zur Laufzeit nicht in den Speicher geladen werden, dazu zählen auch Segmentauslagerungen, interne Overlays und Debugger-Informationen.

Die Größe der .EXE-Datei ist eigentlich nur bei Bestimmung des erforderlichen Speicherbedarfs auf der Festplatte aussagekräftig und hat keinerlei Bedeutung bezüglich des Speicherbedarfs von Overlay-Programmen.

Lademodulgröße, Größe der ausführbaren Datei und Speicher

Tatsächliche Speicheranforderungen

Berechnungen, die zur Link- oder Programmladezeit gemacht werden, können die Speicheranforderungen des Programms und der zu diesem Zeipunkt initialisierten statischen Daten nur schätzen. Alle Hochsprachen bieten die Fähigkeit der dynamischen Speicherbelegung während der Programmausführung, wobei nur die Größe des verfügbaren DOS-Speichers eine Grenze setzt. Daher können kleine Programme, über die eine große Datenmenge verarbeitet wird, zur Laufzeit einen großen

Speicherplatzbedarf haben.

Letztendlich ist die einzige sichere Methode zur Ermittlung des erforderlichen Speicherplatzes eines Programms, den verfügbaren Speicherplatz zu verändern und das Programm jeweils auszuführen.

Code für Overlays auswählen

Der meiste von einem (von Blinker zur Zeit unterstützten) Compiler erzeugte Code ist overlayfähig, vorausgesetzt, es werden far-Aufrufe benutzt. Programmierer, die mit Hochsprachen. Hochsprachen wie C arbeiten, sollten eines der Speichermodelle Medium, Large oder Huge verwenden. Bei CA-Clipper-Anwendungen sind der gesamte CA-Clipper-Code sowie die CA-Bibliothek EXTEND.LIB und Teile der CLIPPER.LIB overlayfähig.

Es ist unvermeidlich, daß beinahe der gesamte Overlay-Code etwas langsamer läuft, als wenn er im Root abgelegt werden würde, weil bei der Handhabung des Overlay-Bereichs immer ein gewisser Systemverwaltungsaufwand besteht, obwohl der Overlaymanager in Assembler geschrieben ist und sehr schnell arbeitet.

Ob dieser Verwaltungsaufwand spürbar ist oder nicht, hängt vom individuellen Programm ab. Aber im allgemeinen sind sehr kleine Routinen und/oder Routinen, die sehr häufig aufgerufen werden, nicht die besten Overlay-Kandidaten, weil der Verwaltungsaufwand des Overlay-Managers im Vergleich zur Laufzeit der Routine riesig sein kann. Solche Routinen beläßt man besser im Root-Bereich.

Ein Beispiel: die Ausführungsgeschwindigkeit eines Programmes, das Grafiken zeichnet, wird sicherlich darunter leiden, daß die setpixel()-Funktion, die von allen Grafikoperationen benötigt wird, ins Overlay gelegt wurde. Denn diese Funktion wird wahrscheinlich mehrere tausendmal in der Sekunde aufgerufen.

Blinker stellt verschiedene Linkzeitbefehle zur Verfügung, wie z.B. BLINKER OVERLAY THRESHOLD und MODULE. Diese helfen Ihnen dabei, den ins Overlay zu legenden Code vom übrigen Code abzusondern. Weitere Einzelheiten zu diesen Befehlen können Sie dem Kapitel 9 entnehmen.

Code für Overlays auswählen

Nicht overlayfähiger Code

Die folgenden Ausnahmen sind besonders für Assembler-Programmierer ineressant:

Interrupt-Handler

Routinen, die Interrupts steuern, KÖNNEN NICHT ins Overlay gelegt werden, da es keine Garantie dafür gibt, daß sie sich im Speicher befinden, wenn der Interrupt aufgerufen wird. Dieser Code MUSS ins Root des Programmes aufgenommen werden.

Daten im Codesegment

Ins Overlay gelegter Code sollte keine Daten in Segmenten verändern oder ablegen, welche der Overlay-Klasse angehören, da diese zu jedwedem Zeitpunkt ein- und ausgelagert werden können, was also einen Datenverlust zur Folge hätte. Außerdem sollte weder im Rootbereich noch im Overlay-Bereich plazierter

Code direkt auf Public-Datenelemente zugreifen, die in einem anderen Overlay-Segment liegen, da Blinker solche Zugriffe nicht erkennen kann. Statt dessen benutzen Sie besser eine Funktion im gleichen Overlaysegment, die den benötigten Wert zurückgibt.

Es ist möglich, temporäre Daten in Codesegmente zu schreiben, aber denken Sie daran, daß ein Aufruf einer Funktion, die innerhalb eines anderen Overlay-Codesegments liegt, dazu führen kann, daß das betroffene Segment aus dem Speicher entfernt wird und die geänderten Daten verloren sind. Um sicher zu gehen, sorgen Sie dafür, daß das gerade ausgeführte Overlaymodul das einzige Overlaymodul im Speicher ist.

Near-Aufrufe

Der Zugang zu einem Overlay-Segment muß über einen far-Aufruf realisiert werden, genau wie ein Aufruf einer Routine vom Overlay ins Root oder in ein anderes Overlay-Segment ebenfalls mittels eines far-Aufrufes erfolgen muß. Rücksprünge aus Overlays müssen ebenfalls vom Typ far sein. Near-Aufrufe können dann verwendet werden, wenn sich beide Funktionen im Root oder im selben Overlay-Segment befinden. Benutzen Sie niemals Sprünge zum Ein- oder Ausstieg in ein Overlay.

Übergabe von Zeigern

Code, der Zeiger auf anderen Code übergibt, wo diese Zeiger nur zur Laufzeit ausgewertet werden, kann nicht ins Overlay gelegt werden, da nicht gewährleistet werden kann, daß ein Codeabschnitt bei jedem Aufruf immer an der gleichen Stelle im Speicher steht. Beachten Sie, daß infolgedessen auch zur Laufzeit erstellte Sprungtabellen durchkreuzt werden. Lediglich Sprungtabellen, die zur Linkzeit erstellt wurden, sind overlayfähig.

Fast alle Hochsprachen generieren Code, der unter Berücksichtigung der obigen Regeln overlayfähig ist. Assembler-Code enthält jedoch häufig Konstruktionen, die ein Overlaying vereiteln.

Ob eine Routine nun overlayfähig ist oder nicht, kann nur durch Versuche unter sorgfältig kontrollierten Bedingungen herausgefunden werden, um eine fehlerfreie Funktionsweise zu gewährleisten.

Code für Overlays auswählen

Einsatz von Bibliotheken

Eine Bibliothek ist einfach eine Datei, die eine Sammlung vorkompilierter Routinen (.OBJ-Dateien) enthält, für gewöhnlich mit einem beigefügten Index. Sie wird durch ein Hilfsprogramm, einen Bibliotheks-Manager wie LIB.EXE von Microsoft, erstellt, der das Hinzufügen, Löschen und Extrahieren von Objektmodulen ermöglicht.

Der Haupt vorteil von Bibliotheken gegenüber Objektdateien ist aus Entwicklersicht, daß aus einer Bibliothek nur die tatsächlich benötigten Module mit in die Applikation eingebunden werden, wohingegen ein Objektmodul immer insgesamt in das Programm einbezogen wird, ob es nun vollständig benötigt wird oder nicht. Die Verwendung von Bibliotheken verringert also die Programmgröße, ohne daß sich der Programmierer darum kümmern müßte, welche Module im Programm benötigt werden.

Normalerweise gibt es drei Arten von Bibliotheken, auf die der Entwickler zurückgreifen kann:

? Compiler-Bibliotheken

? Bibliotheken von Drittanbietern

? Benutzerdefinierte Bibliotheken

Aus zwei Gründen sind Compiler-Bibliotheken oft nicht overlayfähig. Zum einen ist der in diesen Bibliotheken verwendete Code meist hochoptimierter Assembler-Code und benutzt viele 'near calls', welche nicht overlayfähig sind. Zum anderen ist es für gewöhnlich auch nicht erwünscht, oft benutzte Routinen wie diese ins Overlay zu legen, da die Ausführungsgeschwindigkeit stark darunter leiden könnte. Ausnahmen von dieser Regel sind die CA-Clipper Bibliotheken, von denen große, im Large Modell C geschriebene Teile overlayfähig sind.

Bibliotheken von Drittanbietern sind für viele Compiler / Sprachen erhältlich und bieten gewöhnlich Funktionen, die vom Anbieter nicht standardmäßig geliefert werden, wie Grafik- oder Kommunikationsmodule. Der Hauptvorteil beim Einsatz von Add On-Bibliotheken liegt in der großen Ersparnis an Entwicklungszeit bei geringem Kostenaufwand verglichen mit Zeitaufwand, Mühe und Erfahrung, die normalerweise nötig wären, um adäquate Bibliotheken selbst zu entwickeln.

Selbst erstellte Bibliotheken können wirklich alles umfassen, was die Entwickler als für verschiedene Projekte brauchbar ansehen und enthalten oft Code, der den Entwicklern die Pflege eines betriebsinternen Stils oder Standards, z. B. bei Benutzerschnittstellen, erlaubt.

Während des Linklaufs extrahiert Blinker nur die Routinen der Bibliotheken, die das Programm bei seiner Ausführung wirklich braucht.

Code für Overlays auswählen

Reihenfolge der Bibliotheken

Skriptdateien werden in zwei konzeptionellen Durchgängen bearbeitet - zunächst alle FILE-Kommandos, welche die einzubindenden Objektdateien enthalten, und dann die LIB- und SEARCH-Befehle in der Reihenfolge ihrer Einträge in der Skriptdatei.

Die Reihenfolge der Bibliotheken in der Skriptdatei kann SEHR WICHTIG sein, zum Beispiel dann, wenn mehrere Bibliotheken Routinen gleichen Namens enthalten, jedoch einer der Vorzug gegeben werden muß. Es wird diejenige Routine in das Programm gelinkt, die in der zuerst in der Skriptdatei aufgeführten Bibliothek erscheint. Es kann zu unvorhergesehenen Ergebnissen und Fehlern führen, wenn sie nicht die vom Programm erwartete Routine ist.

Wenn innerhalb von Bibliotheken Referenzen auf andere Bibliotheken vorhanden sind, kann eine sorgfältige Konzeption der Bibliotheks-Reihenfolge den zusätzlichen Systemverwaltungsaufwand für das wiederholte Durchsuchen der Bibliotheken nach unaufgelösten Referenzen vermeiden.

Bibliotheken sollten innerhalb der Skriptdatei in der Reihenfolge der Abhängigkeiten aufgelistet sein. Zum Beispiel sollten Bibliotheken, die andere Bibliotheken aufrufen, vor den Bibliotheken aufgeführt werden, auf die sie verweisen. Das heißt in der Regel, daß nicht overlayfähige selbsterstellte - und Drittanbieter-Bibliotheken vor den BEGINAREA-Befehl plaziert werden. Danach folgen die overlayfähigen Bibliotheken zwischen dem BEGINAREA- und ENDAREA-Befehl. In diesem Bereich plazieren Sie zuerst Ihre eigenen Bibliotheken und dann die overlayfähigen Bibliotheken von Drittanbietern. Schließlich sollten die nicht overlayfähigen Compiler-Bibliotheken, wie zum Beispiel die LLIBCE, am Ende der Skriptdatei nach dem ENDAREA-Befehl ins Root plaziert werden.

Denken Sie daran, daß alles, was zwischen die Befehle BEGINAREA und ENDAREA geschrieben wird, ins Overlay und alles außerhalb dieses Bereiches ins Root gelegt wird.

Zum Beispiel:

```
FILE MAIN           # Hauptmodul im Root für
                    # höhere Geschwindigkeit
# Bibliothek mit größter Priorität im Root
LIB HIGHPRIORITY
LIB ROOT            # andere Root-Bibliotheken
BEGINAREA           # Beginn des Overlay-Bereiches
   FILE CODE1          # overlayfähige.OBJ
   FILE CODE2          # overlayfähige .OBJ
   LIB OTHERLIB     # overlayfähige Bibliothek
   LIB 3RDPARTY     # overlayfähige Drittanbieter-Bibliothek
ENDAREA             # Ende des Overlay-Bereiches
LIB LANGLIB         # Compiler-Bibliothek im Root
```

Achtung:

Blinker benötigt keine Overlay-Manager-Dateien oder -Bibliothek, sie werden automatisch während des Linklaufes in die .EXE-Datei eingefügt.

Mehrere BEGINAREAs und ENDAREAs sind in der Regel unnötig, da Blinker den Code entweder als Overlay oder als Nicht-Overlay behandelt. Im folgenden CA-Clipper 5.x-Beispiel ruft jedoch die overlayfähige USERLIB Funktionen aus der nicht overlayfähigen XXX.LIB auf, welche wiederum Funktionen der overlayfähigen EXTEND.LIB ersetzt. In diesem Fall machen mehrere BEGINAREA/ENDAREA-Bereiche durchaus Sinn.

```
OUTPUT MAIN
FILE MAIN           # Hauptmodul
SEARCH ROOTVM       # wird für Overlay der FUNCKY.LIB benötigt
BEGINAREA
   FILE USER
   LIB USERLIB         # (1) overlayfähige Bibliothek, die Funktionen
            # in XXX aufruft
   LIB FUNCKY52     # Hauptbibliotheken FUNCky
   LIB FUNCKYVM
   LIB FUNCKY2C
ENDAREA
LIB XXX                # (2) nicht overlayfähige Bibliothek, die
                    # Funktionen von EXTEND ersetzt und ins
                    # Root gelegt werden muß
BEGINAREA
   LIB EXTEND          # (3) EXTEND folgt und kommt ins Overlay
ENDAREA
LIB CLIPPER         # CLIPPER verbleibt im Root
```

Einige Bibliotheken setzen voraus, daß spezielle Root-Bibliotheken angegeben werden, z.B. ROOTVM.LIB für FUNCky. Einige Bibliotheken erfordern auch die Angabe einer bestimmten .OBJ-Datei im Hauptverzeichnis. In vielen Fällen kann nur eine der aktuelleren Versionen einer Bibliothek ins Overlay gelegt werden. Bitte erkundigen Sie sich deshalb bei Ihrem Händler, ob die von Ihnen genutzte Version overlayfähig ist und klären Sie auch alle weiteren Anwendungsvoraussetzungen.

DOS-Extender- / Dual Mode-Programme

Überblick über DOS-Extender

Programmierung von Protected Mode- und Dual Mode-Anwendungen

Das Extender-API

Linken für den Einsatz im Protected Mode

Linken von DLLs unter dem DOS-Extender

Ausführen von Protected Mode-Programmen

Fehlersuche in Protected Mode-Programmen

Analyse von Allgemeinen Schutzverletzungen (GPFs)

Transparente Interrupt-Unterstützung

DPMI API-Funktionsunterstützung

Windows API-Funktionsunterstützung

Überblick über DOS-Extender

Protected Mode

Der DOS-Extender

Dual Mode

Verwendung von DLLs unter dem DOS-Extender

Speicherverwaltungs-Manager

Auswirkungen auf die Geschwindigkeit

Überblick über DOS-Extender

Protected Mode

Protected Mode ist der erweiterte Modus eines 80286-er oder höheren Prozessors. Diese Prozessoren sind abwärtskompatibel, was von wesentlicher Bedeutung ist. Sie wurden jedoch wegen des Bedarfs an einer Erweiterung des Adressbereiches über 1 MB hinaus entwickelt.

Beim Einschalten des Gerätes starten diese Prozessoren im 8086-er Modus, der auch als Real Mode bezeichnet wird. Wird keine weitere spezielle Software geladen, arbeiten sie weiter wie ein 8086-er, allerdings mit höherer Geschwindigkeit wegen ihrer höheren Taktfrequenz, aber auch aufgrund schnellerer Verarbeitungszeit von Befehlen.

Über spezielle Software-Anweisungen kann der Prozessor jederzeit in den Protected Mode umgeschaltet werden. Dann funktioniert der Prozessor als wesentlich weiterentwickelter Prozessor und besitzt eine Vielzahl zusätzlicher Fähigkeiten, einschließlich der Möglichkeit, auf mehr als 1 MB Hauptspeicher zugreifen zu können. Der Prozessor kann jederzeit zwischen dem Protected Mode und dem Real Mode hin- und herschalten, wenn es erforderlich ist. Der 80286-er Prozessor benötigt dafür jedoch jedesmal ziemlich viel Zeit (Millisekunden), wenn er in den Protected Mode umschalten soll.

Im 80286-er Protected Mode wird ein Segment nicht als absolute Speicheradresse interpretiert, sondern als Index zu einer Adresstabelle, die physische 24-Bit-Basisadressen für alle Speichersegmente im System enthält. Über diese 24-Bit-Basisadresse und ein 16-Bit-Offset ist der Prozessor in der Lage, 16 MB physischen Speicher zu adressieren. Der Speicher ist noch segmentiert, das größte Segment ist weiterhin 64 KB groß.

Überblick über DOS-Extender

Der DOS-Extender

Der 80286-er und weiterentwickelte Prozessoren arbeiten solange im Real Mode, bis anderweitige Anweisungen erfolgen. DOS selbst ist so geschrieben, daß es nur im Real Mode laufen kann. Die Idee zu den DOS-Extendern war, aktuelle Programme im Protected Mode ablaufen zu lassen, ihnen dabei den Zugriff auf sämtlichen zusätzlichen Speicher sowie auf alle Features zu ermöglichen und temporär zum Real Mode zurückzuschalten, wenn ein DOS- oder BIOS-Dienst benötigt wird, der nicht im Protected Mode vorhanden ist.

Ein DOS-Extender besitzt normalerweise einen Codeteil für den Real Mode, der in den konventionellen Speicher geladen wird, und einen Codeteil für den Protected Mode, der in den Erweiterungsspeicher geladen wird. Der Teil für den Real Mode ist für das Laden und die Ausführung des Benutzerprogramms verantwortlich. Er bekommt jedesmal dann die Steuerung, wenn das Programm eine Systemresource anfordert oder auf diese zugreift. Der Protected Mode-Teil enthält gewöhnlich die Ersetzungen (Code) für einige Dienste, die vom DOS oder BIOS angeboten werden. Dadurch ist er in vielen Fällen in der Lage, die aufgerufene Funktion durchzuführen und die Steuerung an das Programm zurückzugeben, ohne daß der Protected Mode verlassen werden muß.

Wenn für die Funktion jedoch DOS oder das BIOS aktiviert werden muß, dann speichert der DOS-

Extender den aktuellen Protected Mode-Status des Prozessors ab, schaltet in den Real Mode um und ruft den für den Real Mode verantwortlichen Programmteil auf. Dieser Teil kann dann selbst die Anfrage erfüllen oder diese auf gewohnte Art und Weise an DOS oder BIOS weitergeben. Wurde die Anweisung ausgeführt, dann geht die Steuerung an den Programmteil für den Real Mode zurück, der leitet die Umschaltung in den Protected Mode in die Wege und gibt danach die Steuerung an den für den Protected Mode verantwortlichen Programmteil zurück. Dieser stellt nun den aktuellen Protected Mode-Status wieder her und fährt mit der Ausführung des Benutzerprogramms fort.

DOS-Extender können verschiedene Features des 80286-er oder 80386-er Prozessors nutzen. Sie nehmen den Prozessor, für den sie geschrieben wurden, nur minimal in Anspruch. Weil die 80x86-er Prozessoren abwärtskompatibel sind, läuft der DOS-Extender in jedem Fall auch auf allen anderen Prozessoren, die weiterentwickelt sind als der, für den das Programm geschrieben wurde.

80286-er DOS-Extender laufen im 16-Bit-Protected Mode, über den der Zugriff auf bis zu 16 MB Speicher in 64-KB-großen Segmenten möglich ist. Weil der Speicher noch in Segmenten à 64 KB verwaltet wird, sind für die meisten Hochsprachen-Programme nur kleine Änderungen nötig, damit sie im 16-Bit-Protected Mode ausgeführt werden können. Normalerweise genügt es, das Programm mit einigen neuen .OBJ- und .LIB-Dateien, die Ersetzungen für bestimmte inkompatible Teile der alten Dateien enthalten, neu zu kompilieren und zu linken. Diese Ersetzungsdateien werden als Teil des DOS-Extenders mitgeliefert und ermöglichen es, daß die Programme im 16-Bit-Protected Mode ausgeführt werden können. Nun kann der Programmierer damit beginnen, die letzten Inkompatibilitäten zu beseitigen.

80386-er DOS-Extender laufen im 32-Bit-Protected Mode, über den der Zugriff auf bis zu 4 GB Speicher in einem oder in mehreren Segmenten ermöglicht wird. Die meisten Programme verwalten den Speicher als einen linearen Adressbereich in einem Segment. Weil der Adressbereich linear ist und Segmente in ihrer Größe nicht mehr auf 64 KB beschränkt sind, treten in bestehenden Programmen wesentlich größere Inkompatibilitäten auf. Für Hochsprachen sind gewöhnlich 32-Bit-Versionen des Compilers erforderlich, um ein Programm für den 32-Bit Protected Mode zu erzeugen. Diese Compiler verhindern die Verwendung der meisten, im Protected Mode nicht erlaubten Konstrukte automatisch.

Der erweiterte Modus von Windows 3.x entspricht derzeit diesem 32-Bit-Protected Mode. Der mit dem Blinker ausgelieferte DOS-Extender läuft auf allen Prozessoren vom 80286-er mit 2 MB Speicher bis hin zum Pentium Pro.

Wegen der Probleme, die potentiell in einer DOS-Umgebung vorhanden sind, in der mehrere Programme laufen, die regelmäßig zwischen Real und Protected Mode hin- und herschalten (insbesondere zwischen EMS-Emulatoren und DOS-Extendern), werden für das Umschalten zwischen Real und Protected Mode zwei unterschiedliche Programmierschnittstellen angeboten. Die erste Schnittstelle heißt Virtual Control Program Interface (VCPI) und wurde speziell für 80386- und 80486-er Prozessoren entwickelt. Die zweite und etwas allgemeinere ist das DOS Protected Mode Interface (DPMI), das in 80286-er und weiter entwickelten Systemen eingesetzt werden kann. Weitere Einzelheiten zu diesen Spezifikationen finden Sie im Anhang C - Literaturhinweise. Dort finden Sie auch Informationen darüber, wie diese Schnittstellen programmiert werden.

Überblick über DOS-Extender

Dual Mode

Die Fähigkeit, Dual Mode-Programme erzeugen zu können, indem man einfach einen einzelnen Befehl in die Link-Skriptdatei einfügt, stellt einen wesentlichen Durchbruch dar. Dual Mode-Programme überprüfen beim Start, ob genügend Systemresourcen vorhanden sind, um das Programm im Protected Mode ausführen zu können. Ist das nicht der Fall, dann wir dasselbe Programm als normales Programm mit dynamischen Overlays ausgeführt.

Die .EXE-Datei enthält nur noch ein Programm, das in beiden Modi ausgeführt werden kann, und nicht zwei gesonderte Kopien, die irgendwie miteinander verbunden sind. Die Vergrößerung der .EXE-Datei gegenüber einem Single Mode-Programm ist vernachlässigbar gering. Auch die Ausführungsgeschwindigkeit ist unbedeutend kleiner, da das gesamte Programm in den Speicher geladen wird, wenn es im Protected Mode ausgeführt wird.

Damit ein Programm im Dual Mode funktioniert und wirklich leistungsfähig ist, muß sämtlicher Code so geschrieben sein, daß er sowohl im Real als auch im Protected Mode ausführbar ist. In Hochsprachen wie CA-Clipper sind keine Codeänderungen erforderlich. Änderungen in C-, PASCAL- und FORTRAN-Programmen stellen die Ausnahme von der Regel dar. Diese Anforderung dürfte nur die C- und ASM-Systemprogrammierer betreffen. Im schlimmsten Fall muß der Code im jeweiligen Prozessormodus getestet, d.h. ausgeführt, werden und danach muß der fehlerauslösende Code verändert werden.

Wir empfehlen C- und Assembler-Programmierern, nicht zwei gesonderte Bibliotheken für den Protected und für den Real Mode zu erzeugen, da bei der Erzeugung eines Dual Mode-Programms beide Codetypen zur Linkzeit eingebunden werden müssen. Vorteile von Dual Mode-Code bzw. der Erzeugung einer Dual Mode-Bibliothek sind:

- Es muß nur eine Kopie des Quelltextes und eine Bibliothek gepflegt werden.
- Beim Versand wird weniger Diskettenplatz benötigt.
- Der Technische Support wird reduziert, da Kunden gar nicht erst in die Lage versetzt werden, mit der falschen Bibliothek zu linken.

Überblick über DOS-Extender

Verwendung von DLLs unter dem DOS-Extender

Um Konzept und Anwendung von DLLs (Dynamic Link Libraries) vollständig zu verstehen, lesen Sie bitte den Abschnitt ‘Dynamisches Linken und DLLs’ im Kapitel 4 des vorliegenden Handbuches. Dieser Abschnitt enthält ausführliche Erklärung über die Funktion des dynamischen Linkens, den Aufbau einer DLL und erläutert die entsprechenden Zusammenhänge für den Programmierer. Auch Vor- und Nachteile der Verwendung von DLLs in einer Applikation werden in diesem Abschnitt diskutiert.

DLLs können nur von Extended Mode-Programmen verwendet werden, der Einsatz unter Dual Mode-Programmen ist nicht möglich, da im Real Mode zu wenig Speicher zur Verfügung steht. Auch ist der Lademechanismus für DLLs nicht kompatibel zum Real Mode. Der Extender lädt Standard-Windows 3.x-DLLs und führt sie aus, da DLLs unter dem DOS-Extender exakt den gleichen Anforderungen und Einschränkungen wie DLLs unter Windows bezüglich dem Codetyp, der in einer DLL enthalten sein darf oder nicht, unterliegen.

Es werden bis zu 32 DLLs je Programm unterstützt, einschließlich der DLLs, die von anderen DLLs aufgerufen werden. Zusätzlich zu der .EXE-Datei wird für jede DLL jeweils ein weiterer File-Handle benötigt, auch wenn auf den Code der DLL nicht zugegriffen wird. Vor Umschalten in den Protected Mode erhöht der DOS-Extender die File-Handle-Tabelle auf 255, d.h. die maximale Anzahl der geöffneten Dateien wird durch die CONFIG.SYS und die Netzwerk-Konfigurationsdateien bestimmt.

Zur Laufzeit sucht der DOS-Extender von der Anwendung angeforderte DLLs in den Verzeichnissen in der nachstehenden Aufstellung in dieser Reihenfolge:

- Im Verzeichnis, von dem das .EXE-Hauptprogramm gestartet wurde (dem Lade-Pfad).
- Im aktuellen Verzeichnis.
- In allen Verzeichnissen, auf die die Umgebungsvariable PATH verweist.

Wurden die DLLs an keiner dieser Stellen gefunden, beendet das Programm mit dem Blinker-Laufzeitfehler 1302. Wurden die DLLs zwar gefunden, enthalten jedoch nicht die korrekte Anzahl und Typen an Export-Definitionen, dann beendet das Programm mit einem Blinker-Laufzeitfehler 1319 oder 1320.

Sobald alle benötigten DLLs geortet wurden, beginnt das Programm mit seiner Ausführung. Segmente einer DLL werden bei Bedarf geladen und zwar bei ihrem ersten Aufruf, ähnlich wie bei der .EXE-Datei. Nach dem Laden eines Segments bleiben die verwendeten Ressourcen bis zum Beenden des Programms aktiv - wird der Speicher einmal knapp, lagert der DOS-Extender die am längsten nicht benutzten Segmente auf die Festplatte aus.

Überblick über DOS-Extender

Speicherverwaltungs-Manager

Programme, die mit dem Blinker DOS-Extender gelinkt wurden, benötigen zumindestens das Programm HIMEM.SYS auf dem Rechner, auf dem sie im Protected Mode betrieben werden sollen. Ist zur Laufzeit kein Speicherverwaltungs-Manager präsent, laufen Dual Mode-Programme im Real Mode ab, Extended Mode-Programme brechen mit folgender Meldung ab:

```
Dieses Programm verwendet einen DOS-Extender und erfordert deshalb mindestens einen
286-Prozessor und außerdem mindestens nnnn KB freien DPMI-, VCPI- oder XMS-Speicher,
um es starten zu können. Sorgen Sie dafür, daß auf Ihrem Rechner ein Speichermanager
installiert ist, der diese Ressourcen zur Verfügung stellt.
```

Der Wert von nnnn ist bestimmt von dem optional als Parameter übergebenen Wert beim Kommando BLINKER EXECUTABLE EXTENDED nnnn in der Link-Skriptdatei. Falls er dort nicht angegeben wurde, beträgt seine Voreinstellung 512 KB.

Alle anderen Speicherverwaltungs-Manager wie QEMM, 386MAX, NetRoom etc. werden voll unterstützt und all deren Fähigkeiten eingesetzt. Mit Blinker für den DOS-Extender gelinkte Anwendungen sind ebenfalls voll kompatibel zur DOS-Box von gehobeneren Betriebssystemen wie Windows 3.x, Windows 95/98, Windows NT, OS/2 und VM/386.

Der Blinker DOS-Extender sucht zuerst nach einem DPMI Host für maximale Kompatibilität , wenngleich ein VCPI oder ein XMS Host einer DOS-Extender-Applikation unter knappen Bedingungen mehr Speicher zur Verfügung stellen kann. Auf Rechnern mit kleinerer Speicherkonfiguration kann es also effektiver sein, nur den HIMEM.SYS zu installieren.

Insbesondere QDPMI benötigt beim Programmstart etwa 70 KB konventionellen Speichers, und dieser Bereich kann nicht durch die SWAPRUNCMD()-Funktion ausgelagert werden. Deshalb wird bei Einbinden des Swappers die QDPMI-Unterstützung automatisch abgeschaltet. Diese Voreinstellung kann jedoch mit dem Link-Skriptkommando BLINKER HOST QDPMI oder dem Umgebungsparameter /HQ erzwungen werden.

Hinweis: Bei Erzeugung eines Menüprogramms zum Swappen in ein DOS-Extender-Programm ist es speichereffizienter, das Menüprogramm als Real Mode-Programm zu erzeugen und die SWPRUNCMD()-Funktion zu verwenden als das Menüprogramm under dem DOS-Extender zu linken.

Überblick über DOS-Extender

Auswirkungen auf die Geschwindigkeit

Programme, die unter dem Blinker DOS-Extender laufen, haben transparenten Zugriff auf bis zu 16 MB direkt adressierbaren Speicher. Speicherintensive Programme, insbesondere solche, die eine große Anzahl an Datenstrukturen verwenden wie z.B. Arrays, laufen deshalb merklich schneller im Protected Mode. Da gewöhnlich der Programmcode nicht in Overlays abgelegt wurde, ist auch die Ausführung des Programms selbst schneller - es muß ja viel weniger auf die Platte zugegriffen werden.

Auf der anderen Seite können Programme, die häufiger auf DOS, BIOS oder externe Ressourcen zugreifen, langsamer unter dem DOS-Extender als im Real Mode ablaufen. Das ist jedoch äußerst programmabhängig. Der zeitliche Aufwand zum Umschalten zwischen Real und Protected Mode variiert je nach Computer, er verringert sich jedoch drastisch beim Aufsteigen von einem 80286er auf einen 80386er, 80486er oder einen Pentium-Prozessor.

Compiler-spezifische Speicherverwaltungen wirken sich manchmal sehr ineffektiv auf die Leistung aus, wenn es darum geht, Megabytes an Daten zu verwalten. Das liegt in erster Linie daran, daß sie eher für die Verwaltung von ein paar hundert KB konzipiert wurden. Hier mag sich die geplante Leistungssteigerung eher negativ auswirken. Aus diesem Grund bietet Blinker ersatzweise eigene Speicherverwaltungs-Funktionen für einige der mehr verbreiteten Compiler.

Programmierung von Protected Mode- und Dual Mode-Anwendungen

Umwandlung eigenen Codes

Spezielle Hinweise zum Protected Mode-Code

Spezielle Hinweise zum Dual Mode-Code

Spezielle Hinweise zum DLL-Code

Spezielle Hinweise zu CA-Clipper-DLL-Code

Low-Level-Programmierung

Programmierung von Protected Mode- und Dual Mode-Anwendungen

Umwandlung eigenen Codes

Beim Konvertieren eines bestehenden Codes ist folgender Punkt wesentlich: Je höher das Niveau der verwendeten Programmiersprache, desto einfacher ist die Umwandlung des Quelltextes. In Hochsprachen wie CA-Clipper sind keine Änderungen erforderlich, was auch immer im Programm erfolgen soll, ist in der eingebundenen CLIPPER.LIB enthalten. Mittelstufige Programmiersprachen wie C, PASCAL oder FORTRAN etc. benötigen in ihrer reinen Form ebenfalls keine Code-Änderungen. Die meisten Änderungen ergeben sich, wenn ein Programmierer Direktzugriffe auf Systemressourcen hart kodiert hat, wie beispielsweise bei der Verwendung eines absoluten Zeigers auf einen Videopuffer.

Solch ein Code muß neu angepaßt werden. Die Zeiger müssen in einer Variablen abgelegt werden, die zu Beginn des Programmes eingerichtet wird. Im Falle von Video-Ausgaben würde beispielsweise in der Startroutine ein Systemaufruf erfolgen, um einen Zeiger auf den Videopuffer einzurichten. Das ist dann entweder ein absoluter Zeiger, wenn das Programm im Real Mode ausgeführt wird, oder ein Selektor, wenn das Programm im Protected Mode läuft. Von da ab laufen unabhängig vom Prozessormodus alle Videospeicherzugriffe über diesen Zeiger, so daß sich kein weiterer Code- oder Zeit-Overhead ergibt.

Assembler-Code bedeutet bei der Umstellung den größten Aufwand, da er Direktzugriff auf Systemresourcen ermöglicht, die im Protected Mode nicht zur Verfügung stehen. Der umzustellende Code ist ähnlicher Natur wie der soeben beschriebene, nur daß es wesentlich wahrscheinlicher ist, daß dies bei Assembler-Code schnell überhand nimmt.

Abschließend müssen die Segmentadressen umgestellt werden. Im Protected Mode werden Segmente nicht sequentiell zugewiesen und sind nicht benachbart. Daher ist der Vergleich von Segmentadressen unzulässig. Eine sequentielle Segmentbelegung beispielsweise erfolgt über den DOS-Extender am günstigsten über Selektoren, die sich durch ein Inkrement von 8 voneinander unterscheiden. Im ungünstigsten Fall werden sie in scheinbar zufälliger Folge produziert. Der Grund dafür liegt darin, daß die Selektoren lediglich Zeiger auf ein Array darstellen, das als Deskriptortabelle bezeichnet wird. Die letzten drei Bits eines Selektors werden zum Speichern der Kontrollinformation verwendet. Weitere Einzelheiten finden Sie unter dem Eintrag zu der Funtion DosGetHugeShift() in der Online-Hilfe.

Um Ihnen einen Überblick über den Konvertierungsprozeß zu geben, der sich in Ihren Ohren wahrscheinlich wie eine riesige Aufgabe anhört, ein Beispiel: Wir selbst haben erst kürzlich die Aufgabe auf uns genommen, Blinker für den Dual Mode umzustellen. Blinker besteht derzeit aus etwa 80.000 Zeilen (5 MB) an reinem Assembler-Code, was ja wohl der am schwersten umstellbare Code ist. Die gesamte Konvertierung auf Dual Mode, d.h. lauffähig im normalen DOS Mode ODER im Protected Mode unter unserem eigenen DOS-Extender, erledigte ein Programmierer in weniger als einem Tag. Der Code ist nun ohne bedingt kompilierten Code zu beiden Modi vollständig kompatibel. Der zusätzliche Overhead für verschiedene Initialisierungen zu Beginn des Programms beträgt weniger als 100 Zeilen.

Im Protected Mode findet man über den Debugger den zu konvertierenden Code relativ leicht: Starten Sie einfach das Programm und schauen Sie, wo ein Fehler auftritt. Normalerweise wird eine Fehlermeldung ausgegeben, in der Fundort und Art des Fehlers beschrieben sind. Danach gibt das Programm die Steuerung an DOS zurück. Unter Windows oder OS/2 kann dies viele Male erfolgen, ohne daß ein Neustart des Rechners erforderlich wäre.

Programmierung von Protected Mode- und Dual Mode-Anwendungen

Spezielle Hinweise zum Protected Mode-Code

Inkompatibilitäten zum Protected Mode treten gewöhnlich durch verschiedene Beschränkungen auf, die im Prozessor eingebaut sind und daher von ihm selbst hervorgerufen werden. Solche Beschränkungen bieten einen besseren Schutz und eine günstigere Abschirmung zwischen verschiedenen Programmen, die auf demselben Prozessor ausgeführt werden. In Hochsprachen-Compilern wird ein Code, in dem diese Beschränkungen nicht eingehalten sind, oft sehr schnell gefunden. Es gibt jedoch auch andere unerlaubte Konstrukte, die erst zur Laufzeit offenbar werden. Folgende Beschränkungen in den Programmiertechniken sind zu beachten:

- Versuche, auf Adressen außerhalb (d.h., hinter dem Ende) des gültigen Segments zuzugreifen, führen zu einer Schutzverletzung.
- Versuche, Code während der Ausführung direkt zu modifizieren oder Daten direkt auszuführen, führen zu einer Schutzverletzung.
- Die Verwendung von Segmentregistern für alle anderen Zwecke außer als Segmentregister. Manchmal werden Segmentregister durch Code zum temporären Speichern von Daten mißbraucht, anstatt die Daten in den Speicher zu schreiben. Wird der Wert eines Segmentregisters modifiziert, muß er einen gültigen Selektor enthalten, sonst tritt eine Schutzverletzung auf.
- Die Durchführung von Real Mode-Adressarithmetik, wie z.B. eine Zeigernormalisierung, ist nicht gestattet, da die Segmente nicht mehr fortlaufend oder benachbart sind. Dies geschieht normalerweise, wenn Datenelemente, die größer als 64 KB sind, verwaltet werden. Solcher Code muß neu programmiert werden.
- Alle Zeiger auf Datenpuffer, die an Real Mode-Code übergeben werden (wenn z.B. ein Software-Interrupt ausgeführt wird oder eine Übertragung zu einem speicherresidenten Programm stattfindet), müssen eine gültige Real Mode-Adresse enthalten. Daher muß der Speicher für den Pufferteil des realen Speichers unterhalb 1 MB sein. Die einzige Ausnahme dazu bilden bestimmte DOS und BIOS Funktionsaufrufe, die transparent verwaltet werden. Eine vollständige Aufstellung von ihnen finden Sie am Ende dieses Kapitels.

Es muß noch einmal nachdrücklich betont werden, daß diese Programmkonstrukte definitiv die

Ausnahme von der Regel darstellen. In Hochsprachen ist es wenig wahrscheinlich, daß solche Konstrukte auftreten.

Der DOS-Extender unterstützt jedoch alle Aufrufe von Laufzeitbibliotheks-Funktionen, einschließlich der EXEC-Funktionen, die den Aufruf eines Programms von einem anderen ermöglichen, mit Ausnahme der ‘chain’-Funktion, die das aufrufende Programm mit neuem Programmcode überschreibt.

Programmierung von Protected Mode- und Dual Mode-Anwendungen

Spezielle Hinweise zum Dual Mode-Code

In diesem Abschnitt finden Sie Empfehlungen dazu, wie Code am günstigsten angepaßt wird, der sowohl im Real als auch im Protected Mode lauffähig sein soll, anstatt auf einen dieser Modi festgelegt zu sein.

Beim Linken von DOS-Extender- oder Dual-Mode-Programmen ist es immer erforderlich, eine zusätzliche, von Blinker für den jeweiligen Compiler mitgelieferte Bibliothek einzufügen. Diese Bibliothek enthält die .OBJ-Ersetzungen für alle .OBJs der entsprechenden Compiler-Bibliothek, die nicht Protected Mode-kompatibel sind.

Alle Blinker-Bibliotheken besitzen Bezeichnungen in der Form BLXcccnn, wobei ccc auf die Firma sowie auf den Compiler und nn auf die Versionsnummer des Compilers verweist. Sie wurden alle im Large Model kompiliert und sollten daher auch nur mit Large Model-Programmen verwendet werden. Der Abschnitt ‘Linken für den Einsatz im Protected Mode’ weiter hinten in diesem Kapitel zeigt weitere detaillierte Informationen.

Wie bereits gesagt, der größte Teil des Codes wird in beiden Modi ohne Modifikationen laufen. Der größte Teil des Rests läuft in einem der beiden Modi, nachdem er so verändert wurde, daß beim Start eine Variable mit dem entsprechenden Wert eingerichtet wird. Im Falle der direkten Adressierung des Videopuffers in C könnte beispielsweise folgender Code verwendet werden:

```
#include <stdio.h>
#include <blx286.h>

BYTE ProcMode;
static SEL Selector;
static unsigned char far *VideoBuffer;
if (DosGetMachineMode(&ProcMode) == 0)
   switch (ProcMode)
      {
      case 0:
         printf ("\nReal Mode - absolute Zeiger ok\n");
         VideoBuffer = MK_FP(0xb800,0);
         break;
      case 1:
         printf ("\n286 Protected Mode - viel Speicher\n");
         if (DosMapRealSeg (0xb800, 4000l, &Selector))
            {
            printf ("\nProblem\n");
            exit (1);
            }
         VideoBuffer = MK_FP(Selector,0);
         break;
      default:
         printf ("\nAnderer Mode - interessant\n");
         exit(1);
```

```
    }
```

Dann:

```
*VideoBuffer[0] = '*';     /* Zeigt in der oberen linken Ecke
                           einen '*' an */
```

Wie Sie sehen konnten, ergibt sich kein zusätzlicher Overhead für Code und keine Verringerung der Ausführungszeit, nachdem die Initialisierung vorgenommen wurde.

Für den Ausnahmefall, daß sich der Code für die einzelnen Modi total voneinander unterscheidet, kann eine der beiden folgenden Methoden verwendet werden. Die erste besteht in der einfachen Nutzung einer 'if' - oder 'switch' - Anweisung über eine globale Variable, wie es im obigen Beispiel für den Initialisierungscode gezeigt wurde. Wenn die Auswahl jedoch über den Code erfolgen soll, dann ist es günstiger, die zweite Methode - die Verwendung eines Funktionszeigers - einzusetzen.

```
#include <stdio.h>
#include <blx286.h>

int (*DoCode) (int);    /* Variable für den Funktionszeiger */
int ProtCode (int);     /* Funktion für Protected Mode-Code */
int RealCode (int);     /* Funktion für Real Mode-Code */

if (ProcMode)       /* Wenn im Protected Mode */
   DoCode = ProtCode;
else                /* Real Mode */
   DoCode = RealCode;
.....

int ProtCode (int abc)
{
   printf ("ProtCode %d\n", abc);
   return (abc * 2);
}
.....
```

Dann:

```
DoCode(1234);       /* Entsprechenden Code ausführen */
```

Beachten Sie bitte, daß in den wenigen Fällen, wo ein Overhead erforderlich ist, dieser absolut minimal ausfällt. Auf einem Rechner mit einem 80286-er Prozessor beträgt er in C weniger als ein Millionstel einer Sekunde. Im Falle einer Hochsprache wie CA-Clipper (die für jede 'if'-Klausel im CA-Clipper-Quelltext hundert Tests durchführt) ist der relative Overhead so klein, daß er total unwesentlich ist.

Es ist eine allgemein verbreitete irrtümliche Annahme, daß dieser Konstrukttyp ein in einer Hochsprache wie CA-Clipper geschriebenes Programm verlangsamt. Jeder, der diese Dinge einmal mit einem Assemblersprachen-Debugger verfolgt hat, wird wissen, daß gerade die Übergabe einer einfachen logischen oder Integer-Variable von CA-Clipper nach C oder ASM hunderte CPU-Anweisungen, einschließlich einer großen Anzahl von Test- und Sprunganweisungen wie der oben angegebenen benötigt.

Die Größe des Overheads ist nur unbestimmt, wenn der Code hunderttausende Male ausgeführt wird, ohne daß eine Beeinflussung stattfindet, d.h., beim Aufrufen von oder Zurückspringen in eine Hochsprache.

Programmierung von Protected Mode- und Dual Mode-Anwendungen

Spezielle Hinweise zum DLL-Code

Der Abschnitt ‘Dynamisches Linken und DLLs’ in Kapitel 4 beschreibt die Anforderungen an die Programmierung und wie Programm-Code und Daten in einem Windows-DLL abgelegt werden sollen. Compiler, die Windows und DLLs unterstützen, enthalten typischerweise einen Abschnitt, in dem die Voraussetzungen für die Windows-Programmierung aufgeführt sind. Genauere Informationen zu Ihrem Compiler entnehmen Sie bitte Ihrem Handbuch.

Es ist besonders wichtig, den Code mit den richtigen Compiler-Optionen zur Erzeugung einer DLL zu übersetzen, da andernfalls Allgemeine Schutzverletzungen (General Protection Faults, auch GPFs genannt) zur Laufzeit verursacht werden können. Programmcode, der in Clipper geschrieben ist, benötigt keine speziellen Optionen, damit er in einer DLL abgelegt werden kann.

Auch ist es wichtig, daran zu denken, daß die Schreibweise von importierten und exportierten Symbolen exakt übereinstimmen muß, wie es auch für eine Windows-Kompatibilität erforderlich ist, daß alle importierten und exportierten Symbole in Großbuchstaben angegeben werden. Standardmäßig werden alle Symbole zur Linkzeit in Großbuchstaben umgewandelt. Bei Hochsprachen wie CA-Clipper muß dies nicht getan werden, da dieser Symbole immer in Großbuchstaben umwandelt.

Wenn Sie die Verwendung von gemischter Groß/Kleinschreibung bevorzugen oder benötigen, kann der Befehlt MIXCASE im Link-Skript eingefügt werden. Dann muß jedoch dafür gesorgt werden, daß importierte und exportierte Symbole in Großschrift umgewandelt werden. Hierzu stehen zwei Möglichkeiten zur Verfügung. Die erste und transparenteste ist, exportierte Funktionen immer als PASCAL-Typ zu deklarieren, was unter Windows der Standard-Typ ist, da der Compiler Funktionen des Typs PASCAL bei der Kompilierung immer in Großbuchstaben umwandelt. Die zweite ist, die Funktionsnamen einfach in Großbuchstaben zu deklarieren und zu kodieren.

In Kleinbuchstaben importierte und exportierte Symbole verursachen immer 'Unaufgelöstes EXTERNAL'-Fehler beim Linklauf oder 'Unaufgelöstes IMPORT'-Fehler beim Laden des Programmes. In beiden Fällen ist eine Ausführung des Programmes nicht möglich.

Ein eingeschränkter Teil des Windows-Applikations-Programmierungs-Interface (API) wird vom Blinker DOS-Extender unterstützt, um den Laufzeitbibliotheken zu ermöglichen, sich zu initialisieren und in gleicher Weise wie unter Windows zu arbeiten. Die unterstützten Windows-API-Funktionen enthalten die Speicherallozierungsbefehle LocalXXXXX und GlobalXXXXX, DLL-Lade- und Entladefunktionen sowie bestimmte Startup-Funktionen. Eine Liste der unterstützten Funktionen finden Sie weiter hinten in diesem Kapitel im Abschnitt 'Windows API-Funktionsunterstützung'.

Wird eine dynamisch geladene DLL sowohl Windows- als auch DOS-Extender-Programmen zur Verfügung gestellt, darf sie keine andere Windows-API-Funktion aufrufen (weder direkt noch indirekt), wenn sie auch unter dem DOS-Extender betrieben werden soll. Die Beispielprogramme im Verzeichnis \C\DLL\DYN verwenden die Windows-Funktionen LoadLibrary(), GetProcAddress() und FreeLibrary(), um die Verwendung von statisch und dynamisch gelinkten DLLs in DOS-Extender-Programmen zu illustrieren.

CA-Clipper DLLs

Für Hochsprachen wie CA-Clipper, wurde die selbe Funktionalität in vier Blinker API-Funktionen implementiert: BLILIBLOD(), BLIFUNHAN(), BLIFUNCAL() und BLILIBFRE(). Die beiden Funktionen BLILIBLOD() und BLILIBFRE() verweisen direkt auf die äquivalenten Windows-API-

Funktionen LoadLibrary() und FreeLibrary(), wie im Abschnitt 'Dynamisches Linken und DLLs' in Kapitel 4 beschrieben.

Hochsprachen ermöglichen dem Programmierer in der Regel keinen direkten Zugriff auf Funktionszeiger, so müssen zwei Schritte durchgeführt werden, um eine dynamisch gelinkte Funktion aufzurufen. Der erste Schritt ist es, einen Handle für diese Funktion zu bekommen, gleichermaßen wie für die DLL. Hierzu wird der DLL-Handle und der Funktionsname an die Blinker-Funktion BLIFUNHAN() übergeben. Wenn die angegebene Funktion in der DLL existiert, wird der Handle für diese Funktion zurückgegeben, andernfalls der Wert 0, der anzeigt, daß die Funktion nicht existiert oder auch die DLL einen Funktionsexport anhand des Namens nicht zuläßt.

Im zweiten Schritt , dem tatsächlichen Funktionsaufruf, werden alle Parameter, die von der Funktion benötigt werden, zusätzlich der Handle als letzter Parameter, an die Funktion BLIFUNCAL() übergeben. Diese Funktion führt den importierten Programmcode der Funktion incl. Übergabe der Parameter aus und liefert den Rückgabewert der importierten Funktion zurück. Der entsprechende Teil des Hauptprogrammes sieht wie folgt aus:

```
libhan = BLILIBLOD (dllnme)      // DLL dynamisch laden
if libhan > 32                   // Wenn erfolgreich geladen
   funhan = BLIFUNHAN (libhan,funnme)   // Handle holen
   if funhan > 0                 // Wenn Funktion gefunden
                        // Aufruf mit (mehreren) Parametern
      funres = BLIFUNCAL (funpa1,funpa2,funhan)
                        // Achtung! Funktionshandle zuletzt
   else
      ? "DLL-Datei", dllnme,
         "enthält nicht die Funktion ", funnme
      ?
   endif
   ? "Funktion ", funnme, "liefert ",
      funres                     // Ergebnis anzeigen
   ?
   BLILIBFRE (libhan)            // Bibliothek freigeben
else
   ? "DLL-Datei ", dllnme, "nicht gefunden oder Ladefehler "
   ?
endif
```

Eine alternative Syntax, ohne Aufruf der Funktion BLIFUNCAL(), ist im Beispielprogramm im Verzeichnis \CLP\DLL\DYN auskommentiert. Diese Syntax wird nicht empfohlen, da die Ausführung viel weniger effizient ist und, falls die Funktion nicht existiert, ein Laufzeitfehler verusacht wird.

```
if libhan > 32                          // Wenn erfolgreich geladen
   funres = &funnme (funpa1,funpa2)
endif                                 // ergibt einen Fehler, wenn
                                      // die Funktion nicht ge-
                                      // funden werden kann.
```

Spezielle Hinweise zu CA-Clipper-DLL-Code

Ist ein Funktionsname sowohl im .EXE-Hauptprogramm als auch in einer DLL vorhanden, wird standardmäßig immer die Funktion in der .EXE-Datei ausgeführt, selbst wenn die Funktion direkt aus der DLL aufgerufen wird. Dieses Verhalten ist im Aufbau von CA-Clipper's Symboltabelle begründet, kann jedoch für dynamisch geladene DLLs durch den Einsatz der Funktion BLILIBOVR() übergangen werden.

Bei einem Aufruf der Funktion BLILIBOVR() mit dem Parameter .T. im CA-Clipper-.EXE-Hauptprogramm übergehen Funktionen in einer dynamisch geladenen DLL gleichnamige Funktionen im .EXE-Programm. Hierdurch kann sogar ERRORSYS veranlaßt werden, nach dem Aufruf einer nicht vorhandenen Funktion eine DLL dynamisch zu laden und danach zu versuchen, die Funktion erneut aufzurufen.

Wenn eine CA-Clipper .DLL dynamisch mit BLILIBLOD() geladen und später mit BLILIBFRE() wieder freigegeben wird, können die Resoucen der .DLL, wie z.B. Speicher und Symbole, nicht mehr vor Ende des Programms wieder freigegeben werden.Das ist normalerweise kein Problem, aber Programme sollten es vermeiden, eine CA-Clipper .DLL wiederholt dynamisch zu laden und wieder freizugeben. Es ist besser, die DLL während der ganzen Applikation geladen zu halten. Natürlich kann man eine DLL entladen, ersetzten und wieder neu laden, wenn nötig. Aber das sollte nicht zu häufig geschehen.

Um die Standardgröße des .EXE-Hauptprogramms bei Verwendung von CA-Clipper-DLLs zu beschränken, wird keine der Extra-Bibliotheken DBFCDX, DBFNDX, DBFNTX und LLIBG automatisch mit eingebunden. Sie können jedoch die Datei CLDLLEXT.PRG im Unterverzeichnis CLP so ändern, kompilieren und in das .EXE-Programm einbinden, daß ein Linken dieser Bibliotheken in das .EXE-Hauptprogramm erzwungen wird.

Mindestens eine Funktion im .EXE-Hauptprogramm muß mit Debugger-Informationen kompiliert werden, um sicherzustellen, daß entsprechende Debugger-Funktionen mit eingebunden werden.

Wollen Sie Produkte von Drittanbietern von einer CA-Clipper-DLL aus einsetzen, werden zwei .OBJ-Dateien ähnlich der mitgelieferten Dateien BLxxCLPnn.OBJ benötigt, über welche die unaufgelösten Externals in der DLL aufgelöst werden und über die veranlasst wird, die benötigten Funktionen in die .EXE-Datei abzulegen. Blinkinc steht mit den Herstellern in Verbindung, damit diese Dateien so schnell wie möglich verfügbar sind.

Die Mechanismen, mit denen Clip4Win und FiveWin C/ASM-DLLs dynamisch laden, können nicht für DLLs mit CA-Clipper-Code verwendet werden, da CA-Clipper-Funktionen keine C/ASM-Parameter übernehmen und auch keine C/ASM-Werte zurückgeben. Sie müssen daher immer die BLcCLPnn.OBJ-Dateien mit dazulinken und die Funktion BLILIBLOD(), wie im Abschnitt vorher beschrieben, einsetzen.

Im Normalfall erfolgt der Aufruf der INIT-Funktionen von CA-Clipper in einer statisch gelinkten DLL erst unmittelbar vor dem ersten Aufruf einer Funktion in der DLL, statt gleichzeitig mit dem Aufruf der INIT-Funktionen im .EXE-Hauptprogramm. Damit diese Funktionen vor dem Hauptprogramm ausgeführt werden, legen Sie einfach den Aufruf einer Dummy-Funktion in der DLL in die erste Zeile Ihrer MAIN-Funktion oder der INIT-Funktion im .EXE-Programm. Dadurch wird die DLL initialisiert, was zur Folge hat, daß alle INIT-Funktionen in der DLL ausgeführt werden.

Programmierung von Protected Mode- und Dual Mode-Anwendungen

Low-Level-Programmierung

Als eine Alternative zu der in der Online-Hilfe beschriebenen Verwendung der API-Funktionen wird ein Teil des DPMI-API durch den Blinker DOS-Extender bereitgestellt, auch wenn kein DPMI-Host verfügbar ist. Dies bedeutet, daß Programme so geschrieben werden können, daß sie direkt auf die DPMI-Funktionen zugreifen können, um Segmente zu belegen oder zu verwalten.. Der Vorteil dieser Art Code ist, daß er auf keinen bestimmten Extender begrenzt ist.

Der Nachteil besteht darin, daß sich die Funktionen normalerweise auf einem niedrigeren Niveau befinden als jene Funktionen, die zu dem Blinker Extender-API mitgeliefert wurden. Dadurch kann es sein, daß von Seiten des Programmierers mehr Arbeit erforderlich ist.

Die unterstützten API-Funktionen werden später in diesem Kapitel aufgeführt.

In einem DOS Extended Programm zwingt das FIXED Attribut (gesetzt im SEGMENTS Modul Definitionsbefehl, siehe Kapitel 4, Windows Programme) den Extender, ein .EXE oder .DLL Segment in den konventionellen Speicher statt in den Extended Speicher zu laden. Dadurch kann sowohl im Real Mode als auch im Protected Mode darauf zugegriffen werden. Konventioneller Speicher ist aber rar, daher sollte diese Option nur dann verwendet werden, wenn sie auch erforderlich ist, z.B. für bimodale Interrupthandler.

Das Extender-API

Namenskonventionen

Aufrufkonventionen

Header-Dateien

Das Extender-API

Namenskonventionen

Allen Funktionen des Extender-API wurden die Buchstaben ‘Dos’ als Präfix vorangestellt, um darauf hinzuweisen, daß sie ein Teil des DOS-Extender-API sind. Sie werden in der alphabetischen Funktionsreferenz in der Online-Hilfe genauer erläutert. Alle diese Funktionen können aus solchen Programmiersprachen aufgerufen werden, in denen die Pascal-Aufrufkonventionen verwendet werden. Eine Ausnahme bilden hierbei verschiedene Hochsprachen wie z.B. CA-Clipper.

Blinker enthält ebenfalls eine generische Get-Status-Funktion BLIMGRSTS(), die in allen Sprachen verwendet werden kann, um bestimmte Status-Informationen über das derzeit ausgeführte Programm zu

erlangen. Diese Informationen stehen bei Programmstart fest, so daß keine Funktion zur Verfügung steht, mit der diese Werte verändert werden können, obwohl sie allesamt über die Umgebungsvariable BLINKER gesteuert werden können.

Da BLIMGRSTS() von allen Sprachen aus aufgerufen werden kann, verwendet diese Funktion die normalen Aufrufkonventionen für die jeweils verwendete Sprache. Sie nimmt einen Integer-Parameter entgegen, über den der Typ der angeforderten Statusinformationen angegeben wird. #defines oder entsprechende Äquivalente sind für die wichtigsten Compiler bzw. Programmiersprachen in den Header-Dateien BLX286.* enthalten.

In den nachfolgenden Erläuterungen und Beispielen werden folgende Begriffe verwendet:

Deskriptor. Eine Struktur, die die Größe, die Basis und die Attribute eines Protected Mode-Segments enthält.

Paragraph. Der Segmentteil einer Real Mode-Adresse. Der Paragraph des Farbadapters ist beispielsweise B800h.

Segment. Die tatsächliche, im Speicher belegte oder abgespeicherte Code- oder Dateneinheit.

Selektor. Der Wert, über den ein Segment identifiziert wird, wenn im Protected Mode gearbeitet wird, d.h., der Wert, der im Protected Mode an die Speicherbelegungsfunktionen übergeben bzw. von diesen zurückgegeben wird.

Das Extender-API

Aufrufkonventionen

Alle API-Funktionen des DOS-Extenders verwenden die Pascal-Aufrufkonvention, außer denen, für die etwas anderes angegeben wurde. Das bedeutet, daß die Funktionsargumente von links nach rechts auf den Stack gelegt werden, und daß die aufrufende Funktion NICHT für das Entfernen der Argumente vom Stack verantwortlich ist.

Für alle API-Funktionen müssen far-Aufrufe verwendet werden. Ebenfalls müssen alle Zeiger vom Typ far sein.

Alle API-Funktionen geben im Register AX einen vorzeichenlosen Integer (short) zurück, über den der Erfolg oder Mißerfolg der Funktion dargestellt wird. Ein Wert Null bedeutet Erfolg und jeder andere Wert stellt einen Fehlercode dar. Alle anderen Register werden beibehalten. Muß die Funktion irgendeinen anderen Wert oder ein anderes Ergebnis zurückgeben, dann übernimmt sie als Parameter einen oder mehrere Zeiger auf Variablen, in denen die entsprechenden Ergebnisse abgelegt werden.

Das Extender-API

Header-Dateien

Die Prototypen für die API-Funktionen und ihre Parameter wurden in Header-Dateien definiert. Diese wurden für die Schnittstelle zu C/C++, Pascal und Modula mitgeliefert und besitzen die Dateinamen

BLX286.*. Sie finden die Dateien in den jeweiligen Unterverzeichnissen.

Die Header-Dateien enthalten weiterhin die Definitionen für alle Datenstrukturen, die für das API erforderlich sind, sowie verschiedene Makros, um deren Nutzung zu vereinfachen.

Linken für den Einsatz im Protected Mode

Soll ein Programm auf die Anwendung im Protected Mode oder im Dual Mode umgestellt werden, dann sind die in der Link-Skriptdatei anfallenden Änderungen minimal. Gewöhnlich müssen lediglich zwei zusätzliche Befehle in die Linkdatei eingefügt werden: einer für die Angabe, daß es sich um ein DOS-Extender-Programm handelt und der andere, um die Blinker-Bibliothek für den verwendeten Compiler einzubinden.

Wenn Sie einfach den Befehl BLINKER EXECUTABLE EXTENDED (BLI EXE EXT) einfügen, dann erzeugt der Blinker ein DOS-Extender-Programm. Ähnlich verläuft die Reaktion auf den Befehl BLINKER EXECUTEABLE DUAL (BLI EXE DUA). Mit diesem Befehl wird dem Blinker mitgeteilt, daß ein Dual Mode-Programm erzeugt werden soll.

Beim Linken eines DOS-Extender- oder Dual Mode-Programmes ist es in jedem Falle erforderlich, eine zusätzliche Blinker-Bibliothek für den jeweiligen Compiler anzugeben. Diese Bibliothek enthält die .OBJ-Ersetzungen für alle .OBJs der Compiler-Bibliothek, die nicht Protected Mode-kompatibel sind.

Alle Blinker-Bibliotheken besitzen Bezeichnungen in der Form BLXcccnn, wobei ccc auf die Firma sowie den Compiler und nn auf die Versionsnummer des Compilers verweist. Sie wurden alle im Large Model kompiliert und sollten daher auch nur mit Large Model-Programmen verwendet werden.

Die entsprechende Compiler-Bibliothek muß über den Befehl SEARCH genau vor der Standard-Compiler-Bibliothek oder vor der Skriptdatei angegeben werden, in der die Compiler-Bibliothek enthalten ist.

Beispiel für Microsoft Visual C++:

```
SEARCH BLXMVC15  # Blinker-Bibliothek
LIB LLIBCE          # MSVC 1.5-Bibliothek
```

Beispiel für CA-Clipper 5.3:

```
SEARCH BLXCLP53  # Blinker-Bibliothek
LIB CLIPPER         # CA-Clipper 5.3-Bibliothek
```

oder:

```
SEARCH BLXCLP53  # Blinker-Bibliothek
@CL530MID        # Skript mit der CA-Clipper-Bibliothek
```

Wird die Compiler-Bibliothek nicht explizit erwähnt, dann sollten Sie den SEARCH-Befehl mit der Blinker-Bibliothek in der Linkdatei als letzte Bibliothek angeben.

Werden Bibliotheken von Drittanbietern verwendet, dann sollten Sie vorher überprüfen, ob diese im Protected Mode oder im Dual Mode eingesetzt werden können, bevor Sie versuchen, diese mit einer DOS-Extender-Anwendung zu nutzen. Blinkinc kann nicht dafür garantieren, daß alle Bibliotheken entweder overlayfähig sind oder im Protected Mode eingesetzt werden können oder gar beides. Im Zweifelsfall wenden Sie sich bitte an den Hersteller oder Verkäufer des entsprechenden Produktes.

Die Bibliotheken von Drittanbietern sollten normalerweise vor dem SEARCH-Befehl mit der entsprechenden Blinker-Bibliothek angegeben werden, wenn in dem jeweiligen Add-On-Produkt keine

anderweitigen Angaben gemacht wurden.

Hinweis: Alle Befehle und Funktionen des Overlay-Managers sind für ein Dual Mode-Programm, das im Real Mode läuft, anwendbar. Legen Sie deshalb die Größe des Overlay Caches fest, schalten Sie den OVERLAY PAGEFRAME ein, wenn möglich, und räumen Sie ausreichenden Speicher für den Overlaybereich ein etc.

Linken von DLLs unter dem DOS-Extender

Der DOS-Extender von Blinker lädt Standard-Windows 3.x-DLLs und führt sie aus, daher werden DLLs für Programme unter dem DOS-Extender in genau gleicher Weise erstellt wie DLLs für Windows. Dies bedeutet, daß DLLs gewöhnlich mit der Windows-DLL-Version der Laufzeitbibliothek des verwendeten Compilers gelinkt werden, statt mit der Bibliothek BLXcccnn und der Standard-Laufzeitbibliothek. Die einzige Ausnahme sind DLLs von niedrigen Sprachen wie Pascal oder C/C++, die nur im DOS Extended Mode verwendet werden. Mehr darüber weiter unten.

Wie bereits erwähnt, werden einige Windows API-Funktionen vom Blinker DOS-Extender unterstützt, um den Laufzeitbibliotheken zu ermöglichen, sich zu initialisieren und in gleicher Weise zu funktionieren wie unter Windows. Das heißt aber auch, daß durch eine intelligente Programmierung einer DLL, d.h. keine Aufrufe von Funktionen der DOS-Extender-API- oder der Windows-API-Schnittstelle, die gleiche DLL für DOS-Extender- und für Windows-Programme verwendet werden kann.

So kann beispielsweise eine DLL erzeugt werden, die allen Programmcode enthält, der für die Verwaltung einer Datenbank zuständig ist, mit ein paar exportierten Funktionen um aktuelle Datensätze zu übergeben und so weiter. Sowohl die Windows- als auch die DOS-Version des Hauptprogrammes, sogar mehrere verschiedene Programme können diese Datenbank-DLL verwenden, ohne Programmcode innerhalb der Hauptprogramme doppelt erstellen zu müssen.

Das Unterverzeichnis \DLL im entsprechenden Sprach-Unterverzeichnis des Blinker-Installationsverzeichnisses enthält Beispielprogramme, in denen die Erzeugung von statisch gelinkten und dynamisch gelinkten DLLs veranschaulicht wird. Das Verzeichnis \STC enthält alle Dateien zur Erzeugung einer statischen DLL, im Verzeichnis \DYN kann eine dynamische DLL erzeugt werden. Die C-Beispielprogramme enthalten auch ein Windowsprogramm, mit dem die gemeinsame Verwendung einer DLL unter Windows und DOS gezeigt wird.

Bei Erzeugung einer DLL mit einer Programmiersprache niedrigeren Niveaus, wie C oder Pascal, stehen bestimmte Bereiche der Laufzeitbibliothek, wie z.B. die Funktion printf(), nicht zur Verfügung. Damit nun bestehender Code, der diese Funktionen verwendet, dennoch in DLLs abgelegt wird, ist es auch möglich, DLLs speziell für den DOS-Extender zu erzeugen. Das bedeutet jedoch, daß diese DLLs nicht mehr mit Windows-Programmen betrieben werden können, dafür aber auf alle DOS-Laufzeitbibliotheksfunktionen zugreifen können, die unter Windows nicht zur Verfügung stehen. Das Verzeichnis \C\DLL\EXT enthält ein Programm, mit dem diese Besonderheit dargestellt wird.

Unaufgelöste Externals (Unresolved Externals) werden zur Linkzeit häufig angezeigt, wenn DLLs aus existierendem Code erzeugt werden, da dieser Code Referenzen auf anderen Code beinhalten kann, der sich in anderen Modulen oder im .EXE-Hauptprogramm befindet. Deshalb müssen für alle unaufgelösten Externals EXPORT- und IMPORT-Definitionen angegeben werden, damit das Programm erfolgreich gelinkt und ausgeführt werden kann. Beachten Sie, daß durch die Art, wie Windows-DLLs, Exporte und Importe gestaltet wurden, es nicht einfach ist, Funktionen aus dem .EXE-Hauptprogramm zu importieren, während der Import des .EXE-Hauptprogramms aus der DLL wie auch Export und Import unter DLLs gleichermaßen durchgeführt werden kann.

Die Erzeugung von DLLs mit CA-Clipper erfordert eine spezielle Behandlung, da der Compiler Windows und DLLs nicht ohne weiteres unterstützt. CA-Clipper-Programmierer sollten deshalb das folgende allgemeine Beispiel überspringen und dem Beispiel für CA-Clipper folgen, das unmittelbar danach aufgeführt ist.

In beiden Fällen sehen Sie bitte in der Windows Online-Hilfedatei BLINKER.HLP die letzten Informationen zur Erzeugung von DLLs und zu Einzelheiten, wie der zusätzliche Overhead zum Linken des .EXE-Hauptprogramms minimiert werden kann.

Allgemeines Beispiel einer DLL für Windows und den DOS-Extender

Allgemeines Beispiel einer DLL nur für den DOS-Extender

Beispiel einer DLL für den DOS-Extender unter CA-Clipper 5.x

Linken von DLLs unter dem DOS-Extender

Allgemeines Beispiel einer DLL für Windows und den DOS-Extender

Das Beispielprogramm im Unterverzeichnis \C\DLL\STC des Blinker-Installationsverzeichnisses zeigt, wie ein Windows-Programm und ein DOS-Extender-Programm erzeugt werden, die beide die gleiche Windows-DLL laden und ausführen. Sehen Sie sich die Make-Datei für mehr Einzelheiten an. Für Microsoft Visual C++ 1.5 beispielsweise müßte der Befehl zum Kompilieren folgendermaßen aussehen:

```
cl -c -AL t.c          Normaler Compileraufruf
cl -c -AL -GD d.c         Compileraufruf für eine DLL
```

Nun erzeugen Sie eine eigene Link-Skriptdatei für die DLL, die eine Export-Definition für die zu exportierende Funktion enthält, und linken Sie die DLL unter Verwendung der Windows DLL Laufzeitbibliotheken, in diesem Fall LDLLCEW und LIBW.

Obwohl die meisten Compiler automatisch eine Export-Definition in der .OBJ-Datei ablegen, wenn eine Funktion mit dem Schlüsselwort __export deklariert ist, gibt eine Angabe des Funktionsnamens im EXPORTS Abschnitt der DLL Moduldefinitionsdatei dem Programmierer mehr Einfluß darüber, wie das Symbol exportiert wird. Verwenden Sie nun einen Import-Bibliothekmanager wie z.B. BLILIB (siehe den Abschnitt 'Der BLILIB Import-Bibliotheksmanager' in Kapitel 4), um eine Importbibliothek zu erzeugen, die mit dem aufrufenden Programm gelinkt werden kann:

```
# DW.LNK zur Erzeugung von D.DLL
OUTPUT d                    # Name der DLL-Datei
FILE d                      # Liste OBJ-Dateien hier
DEFBEGIN                    # Modul-Definitionsbefehle
   LIBRARY   'd'               # Zeigt eine DLL und ihren Namen
   EXETYPE   WINDOWS 3.1           # Ist für Windows 3.1 und höher
   HEAPSIZE  1024           # Lokaler Heap von 1 KB benötigt
   EXPORTS                     # Explizite Export-Definition
      _test                 # Export von _test über Namen
DEFEND                      # Windows DLL Definitionsdatei
LIB LDLLCEW                    # Windows DLL Laufzeitbibliothek
LIB LIBW                    # Windows Importbibliothek
NODEF                       # keine Standardbibliotheken

BLINKER @dw                    # die .DLL linken
```

```
BLILIB /ID d /OL d                # Importbibliothek erstellen
```

Schließlich erzeugen Sie die Haupt-Link-Skripdatei, um nun die Import-Bibliothek anstelle der Datei LIMERICK.OBJ zu verwenden:

```
# T.LNK zur Erzeugung von T.EXE
BLINKER EXECUTABLE EXTENDED       # Erzeugt ein DOS-Extender-Programm
OUTPUT t                        # Name der Ausgabedatei
FILE t                          # Hauptprogramm
LIB D                           # soeben erstellte Importbibliothek
LIB BLXMVC15                    # Blinker Protected Mode-Bibliothek

BLINKER @T                         die .EXE Datei linken
```

Nun haben Sie ein Programm T.EXE erzeugt, das Programmcode aus D.DLL lädt und ausführt. Ein ähnliches Beispiel zum dynamischen Linken und Laden von DLLs finden Sie auch im Unterverzeichnis \C\DLL\DYN.

Linken von DLLs unter dem DOS-Extender

Allgemeines Beispiel einer DLL nur für den DOS-Extender

Die Beispielprogramme im Unterverzeichnis \C\DLL\EXT des Blinker-Installationsverzeichnisses zeigen, wie ein DOS-Extender-Programm erzeugt wird, das eine nur unter dem DOS-Extender lauffähige DLL lädt, die einen Aufruf der printf() Laufzeitbibliotheks-Funktion durchführt. Sehen Sie bitte in der Make-Datei für weitere Einzelheiten nach, für die Verwendung mit Microsoft Visual C++ 1.5 würden die Zeilen für den Compileraufruf beispielsweise so aussehen:

```
CL -c -AL t.c          Normaler Compileraufruf
                       für das Hauptprogramm
CL -c -AL -GD d.c         Compileraufruf für eine DLL
```

Nun erzeugen Sie eine eigene Link-Skriptdatei für die DLL, die eine Export-Definition für die zu exportierende Funktion enthält, und linken Sie die DLL mit den gleichen Bibliotheken wie das DOS-Extender-Hauptprogramm, in diesem Fall BLXMVX15 und LLIBCE.

Wie zuvor erklärt, gibt die explizite Deklaration eines Symbols als exportiert dem Programmierer mehr Kontrolle über die Art, wie das Symbol exportiert wird. Nun verwenden Sie eine Import-Bibliotheksmanager wie z.B. BLILIB (siehe den Abschnitt 'Der BLILIB Import-Bibliotheksmanager' in Kapitel 4) um eine Importbibliothek zu erzeugen, die mit dem aufrufenden Programm gelinkt werden kann:

```
# D.LNK zur Erzeugung von D.DLL
BLINKER EXECUTABLE EXTENDED       # Erzeugt ein DOS-Extender-Programm
OUTPUT d                        # Ausgabe-Dateiname
FILE d                          # Liste OBJ-Dateien hier
DEFBEGIN                        # Moduldefinitionsbefehle
   LIBRARY   'd'                   # Zeigt eine DLL und ihren Namen an
   EXPORTS                         # Exportiere folgende Symbole
      _test                     # Exportiere _test über Namen
DEFEND                          # Windows DLL Definitionsdatei
LIB BLXMVC15                    # Blinker Protected Mode-Bibliothek

BLINKER @d                         .DLL linken
BLILIB /ID d /OL d /NP             Importbibliothek erzeugen
```

Schließlich erzeugen Sie noch die Linkdatei für das Hauptprogramm, damit es die gerade erzeugte Importbibliothek anstelle der Datei D.OBJ direkt verwendet:

```
# T.LNK zur Erzeugung von T.EXE
BLINKER EXECUTABLE EXTENDED      # Erzeugt ein DOS-Extender-Programm
OUTPUT t                       # Name der Ausgabedatei
FILE t                         # Hauptprogramm
LIB D                          # Importbibliothek
LIB BLXMVC15                   # Blinker Protected Mode-Bibliothek

BLINKER @T                        .EXE linken
```

Nun wurde eine Datei T.EXE erzeugt, die den Code in D.DLL lädt und ausführt, auch wenn D.DLL nicht unter Windows verwendet werden kann.

Linken von DLLs unter dem DOS-Extender

Beispiel einer DLL für den DOS-Extender unter CA-Clipper 5.x

Aufgrund der Besonderheit der Laufzeitbibliothek von CA-Clipper 5.x, deren Verwendung von Tokens im Code und einer Symboltabelle, benötigt CA-Clipper etwas zusätzlichen Aufwand, damit Programmcode in einer DLL betrieben werden kann, aber das Ergebnis ist eine einzige CA-Clipper-Code DLL für alle Programmtypen.

Das CA-Clipper Laufzeitsystem ist in eine Vielzahl Subsysteme unterteilt, so wie Tastatureingabe, Bildschirmausgabe, Datenbanktreiber etc., die so strukturiert sind, daß nur der jeweilige Code des Subsystems, der vom Programm tatsächlich benötigt wird, in das zu erzeugende .EXE-Programm gelinkt wird. Bei Verwendung einer DLL wäre es dann möglich eine kleine Hauptfunktion zu erzeugen, die einfach eine andere Funktion in einer DLL aufruft, ohne Referenz auf irgendein Subsystem. Wenn nun der tokenisierte Code innerhalb der DLL eines der Subsysteme aufrufen wollte, die ja nicht mitgelinkt wurden, käme es zu einem Laufzeitfehler.

Daher ist es nötig, eine zusätzliche .OBJ-Datei in das .EXE Hauptprogramm zu linken, um sicherzustellen, daß alle Subsysteme im Hauptprogramm enthalten sind. Ebenso wird eine zusätzlichen .OBJ-Datei in jede .DLL-Datei dazugelinkt, die die Schnittstelle zum Hauptprogramm darstellt. Die Dateien BLXCLPnn.OBJ (für die EXE) und BLDCLPnn.OBJ (für jede DLL) - nn steht hier für die Clipper-Versionsnummer - befinden sich im Unterverzeichnis OBJ des Blinkerverzeichnisses. Nach ordnungsgemäßer Installation findet der Blinker diese Dateien automatisch, da die Umgebungsvariable OBJ auch auf dieses Verzeichnis verweist.

Die Protected Mode-Bibliothek BLXCLPnn darf nicht in der Link-Skriptdatei für die DLL enthalten sein, sie wird hier nicht benötigt, ebenfalls nicht die Standardbibliotheken von CA-Clipper, wie CLIPPER, EXTEND, TERMINAL und DBFNTX. Um sicherzustellen, daß dies der Fall ist, überprüfen Sie, ob diese auch wirklich nicht in der DLL-Link-Skriptdatei oder anderen eingebundenen Skriptdateien enthalten sind. Ergänzen Sie Ihre Link-Skriptdatei auch um den Befehl NODEFLIB - so verhindern Sie, daß die Standardbibliotheken in die DLL mitgelinkt werden.

Bedingt durch Laufzeitoperationen der CA-Clipper-Symboltabelle , können unaufgelöste Externals beim Linken einer DLL mit dem Befehl BLINKER EXECUTABLE NODELETE ignoriert und übergangen werden, da diese Funktionen im .EXE-Hauptprogramm oder einer anderen DLL zum Zeitpunkt des Funktionsaufrufs zur Verfügung stehen. Diese Praxis wird für die endgültige Programmversion nicht empfohlen, da Laufzeitfehler verursacht werden, wenn Funktionen nicht zur Verfügung stehen, sie kann

jedoch nützlich sein zum Testen oder für den Aufbau einer DLL.

Um nun z.B. ein Testprogramm mit CA-Clipper 5.3 und eine DLL, die das Beispielsprogramm im Verzeichnis \CLP\DLL\STC enthält, zu erzeugen, kompilieren Sie zuerst die .PRG-Dateien ganz normal.

```
clipper t -m -n         Normaler Compileraufruf
                        für das Hauptprogramm
clipper d -m -n         Normaler Compileraufruf
                        für die DLL
```

Nun erstellen Sie eine eigene Link-Skriptdatei für die DLL, die eine Exportdefinition für die zu exportierende Funktion enthält, und linken Sie danach die DLL. Verwenden Sie danach den Import-Bibliothekmanager BLILIB, um eine Import-Bibliothek zu erzeugen, die wiederum mit dem aufrufenden Programm gelinkt werden kann (siehe Abschnitt 'Der BLILIB Import-Bibliotheksmanager' in Kapitel 4):

```
# D.LNK zur Erzeugung von D.DLL
OUTPUT d                        # Name der Ausgabedatei
FILE d                          # Liste OBJ-Dateien hier
FILE bldclp53                   # .OBJ-Datei für jede .DLL
DEFBEGIN                        # Modul-Definitionsbefehl
   LIBRARY 'd'                     # Zeigt eine DLL an
   EXETYPE WINDOWS 3.1          # Ist für Windows 3.1 und höher
   EXPORTS                         # Exportiere folgende Symbole
      STATTEST                  # Export von STATTEST über Namen
DEFEND
NODEFLIB                        # Verzichte auf Standard-
                                # Bibliotheken

BLINKER @D                         die .DLL linken
BLILIB /ID d /OL d                 Importbibliothek erzeugen
```

Erzeugen Sie nun die Link-Skriptdatei für Ihr Hauptprogramm, um die spezielle .OBJ-Datei und die erzeugte Importbibliothek anstelle der Datei D.OBJ zu verwenden:

```
# T.LNK zur Erzeugung von T.EXE
BLINKER EXECUTABLE EXTENDED        # Erzeugt DOS Extended Mode-Programm
OUTPUT t                        # Name der Ausgabedatei
FILE test                       # Hauptprogramm
FILE blxclp53                   # .OBJ-Datei für .EXE
LIB d                           # Import-Bibliothek
LIB blxclp53                    # Blinker Protected Mode-
                                # Bibliothek

BLINKER @T
```

Nun haben Sie das Programm T.EXE erzeugt, das den Programmcode aus D.DLL lädt und danach ausführt.

Durch Einbinden der Datei BLXCLP53.OBJ in das Hauptprogramm werden alle Laufzeitfunktionen des CA-Clipper in das .EXE-Hauptprogramm eingebunden. Dies hat zur Folge, daß kleine Programme in ihrer Größe wachsen, dafür ist aber sichergestellt, daß jeder CA-Clipper Programmcode in den zum Hauptprogramm gehörenden DLLs abgelegt werden kann. Die Datei CLDLLEXT.PRG im \CLP Unterverzeichnis des Blinker Installationsverzeichnisses kann auch modifiziert, kompiliert und in das Hauptprogramm gelinkt werden, um zu erzwingen, daß einige Teile der Clipper-Laufzeitbibliothek in das .EXE-Hauptprogramm gelinkt werden, wenn benötigt.

Beziehen Sie sich bitte auf die Windows Online-Hilfe-Datei BLINKER.HLP, um letzte Informationen bzgl. der Erzeugung von DLLs zu erhalten und wie der zusätzliche Aufwand beim Linken in das EXE-Hauptprogramm minimiert werden kann.

Ausführen von Protected Mode-Programmen

Protected Mode Laufzeitsystem

Verschachtelter Betrieb mehrerer DOS-Extender-Programme

Link-Skript-Befehle zur Laufzeitsteuerung

Die Umgebungsvariable BLINKER zur Laufzeit

Ausführen von Protected Mode-Programmen

Protected Mode Laufzeitsystem

Wenn der Blinker ein DOS-Extender-Programm erzeugt, dann verändert sich das Format der MAP-Datei. Damit wird die unterschiedliche Programmstruktur verdeutlicht. Die Segmente sind sequentiell durchnumeriert. Dabei wird mit der Segmentnummer 1 begonnen. Viele logische Segmente können in ein einzelnes physisches Segment zusammengelegt werden.

Am einfachsten findet man die Fehler, wenn man das Programm in einer Multitasking-Umgebung ausführt. Insbesondere dann, wenn die Betriebssysteme verschiedene Arten von Task-Isolierung und Maßnahmen zum Schutz vor Abstürzen enthalten, ist es nicht erforderlich, das System jedesmal neu zu starten, wenn eine Schutzverletzung auftritt.

Sobald eine Schutzverletzung oder ein anderer Protected Mode-Fehler auftritt, werden die Inhalte aller Register sowie die mit den Segmentregistern verbundenen .EXE-Segmentnummern standardmäßig auf dem Bildschirm ausgegeben. Diese Segmentnummern sind die gleichen, die auch in der MAP-Datei angegeben werden. Solch eine MAP-Datei wird mit Hilfe der Option MAP S,A erzeugt und kann dazu verwendet werden, um gemeinsam mit den Offset-Registern IP, BX etc. den Fehler und dessen Ursache zu lokalisieren. Weitere Einzelheiten finden Sie weiter unten im Abschnitt ‘Fehlersuche in Protected Mode-Programmen’.

Die Arbeit des DOS-Extenders ist in einer DPMI-Umgebung wesentlich geringer, als in anderen Umgebungen. Daher sind Sie gut beraten, wenn Sie Ihr Programm in all den Umgebungen austesten, von denen angenommen wird, daß das Programm in diesen ablaufen soll. Dazu sollten unbedingt auch DPMI, VCPI und XMS gehören. Wenn sie dem Programmierer nur in der gleichen Umgebung zur Verfügung stehen wie das DPMI, dann kann die DPMI-Unterstützung des Extenders mit Hilfe der Umgebungsvariablen BLINKER (wie unten beschrieben) deaktiviert werden.

Der tatsächliche Modus, in dem der DOS-Extender läuft, kann mit Hilfe der API-Funktion DosGetHostMode() ermittelt werden.

Ausführen von Protected Mode-Programmen

Verschachtelter Betrieb mehrerer DOS-Extender-Programme

DPMI 0.9, wie es in den DOS-Kompatibilitätsboxen von Windows und OS/2 unterstützt wird, besitzt eine wichtige Einschränkung im Hinblick auf das Verschachteln mehrerer DOS-Extender-Programme in derselben Sitzung.

Alle Ressourcen, wie z.B. der Hauptspeicher, die von einem sekundären DPMI-Client belegt werden, d.h. wenn ein DOS-Extender-Programm von einem anderen DOS-Extender-Programm her aufgerufen wird, bleiben zugeordnet, bis der primäre DPMI-Client sein Programm beendet. Zu diesem Zeitpunkt werden alle Ressourcen beider Programme frei und an das System zurückgegeben.

Aus diesem Grund empfehlen wir dringend, bei der Verschachtelung von Programmen mit den Befehlen SPAWN, SHELL oder RUN bzw. mit der BLINKER SWPRUNCMD()-Funktion, das Hauptprogramm als Real Mode-Programm zu konzipieren. Wie bereits an anderer Stelle erläutert, ist dies ein guter Weg, da so der verfügbare Speicher besser genutzt werden kann als bei der Verschachtelung zweier DOS-Extender-Programme.

Ein Beispiel für diese Situation: Sie rufen den letzten R&R Report Writer aus einem DOS-Extender-Programm auf. R&R 6.0 ist ebenfalls ein DOS-Extender-Programm, d.h. der ihm zugeordnete Speicher wird nicht freigegeben, solange das Hauptprogramm nicht beendet ist. Die Segmente werden zwar vom Betriebssystem auf die Platte geswappt, die Selektoren bleiben jedoch solange alloziert, bis das aufrufende Programm beendet wird.

Diese Einschränkung gilt nur für den Betrieb unter einem DPMI 0.9 Host. Bei Programmen, die unter DOS mit einem Speicherverwaltungstreiber EMM386, QEMM386, 386MAX oder NETROOM betrieben werden, tritt dieses Problem nicht auf.

Ausführen von Protected Mode-Programmen

Link-Skriptbefehle zur Laufzeitsteuerung

Das Laufzeitverhalten eines mit Blinker erzeugten DOS-Extender-Programms kann zur Linkzeit über verschiedene Link-Skriptbefehle gesteuert werden. Nachfolgend sehen Sie die aktuellen Befehle und ihre Bedeutung (voreingestellte Werte sind in Fettdruck dargestellt):

BLINKER EXECUTABLE IPX nnn, nnn
Setzt die IPX ECB-Puffer.

BLINKER EXTMEM BLOCK nnn
Setzt die Blockgröße für den Erweiterungsspeicher.

BLINKER EXTMEM LIMIT nnn
Setzt das Maximum des zu nutzenden Erweiterungsspeichers.

BLINKER HOST DPMI ON | OFF
Aktiviert / deaktiviert die Unterstützung für den DPMI-Host.

BLINKER HOST MESSAGE ON | OFF

Schaltet die Anzeige von Statusmeldungen beim Programmstart ein / aus

BLINKER HOST PASSUP ON | OFF
Aktiviert / deaktiviert Weitergabe von Real Mode Interrupts
an Protected Mode

BLINKER HOST QDPMI ON | OFF
Aktiviert / deaktiviert die Unterstützung für den QDPMI-Host.

BLINKER HOST VCPI ON | OFF
Aktiviert / deaktiviert die Unterstützung für den VCPI-Host.

BLINKER HOST XMS ON | OFF
Aktiviert / deaktiviert die Unterstützung für den XMS-Host.

BLINKER LOWMEM RESERVE nnn
Reserviert nnn KB Speicher unterhalb 1 MB für Programme.

Mehr Informationen zu diesen Befehlen finden Sie in der Online-Hilfe.

Ausführen von Protected Mode-Programmen

Die Umgebungsvariable BLINKER zur Laufzeit

Das Laufzeitverhalten eines mit Blinker erzeugten DOS-Extender-Programms kann über die Einstellung der Umgebungsvariablen BLINKER über den DOS-Prompt verändert werden. Diese Einstellungen repräsentieren die Anfangsbuchstaben der entsprechenden Befehle einer Blinker-Link-Skriptdatei.

Im folgenden werden die möglichen Einstellungen und ihre Bedeutung aufgeführt (als Voreinstellung gesetzte Werte sind in Fettdruck dargestellt):

/DTnnnn		Schwellwert für Dual Mode zum Betrieb im Protected Mode überschreiben.
/EBnnnn		Setzt Blockgröße für Erweiterungsspeicher.
/ED+	/ED-	Aktiviert / deaktiviert Extender-Debugging.
/EInnn		Setzt Anzahl der IPX ECB Puffer.
/EK+	/EK-	Aktiviert / deaktiviert spezielle Tastaturbehandlung unter dem DOS-Extender.
/ELnnnn		Setzt maximal zu nutzenden Erweiterungsspeicher
/ETnnnn		Setzt Schwellwert an Erweiterungsspeicher für den Betrieb als EXTENDED Mode-Programm.
/HD+	/HD-	Aktiviert / deaktiviert Unterstützung des DPMI-Host
/HM+	/HM-	Schaltet die Statusmeldungstatus beim Start ein / aus
/HP+	/HP-	Aktiviert / deaktiviert Übergehen von Interrupts
/HQ+	/HQ-	Aktiviert / deaktiviert Unterstützung des QDPMI-Host
/HV+	/HV-	Aktiviert / deaktiviert Unterstützung des

		VCPI-Host
/HX+	/HX-	Aktiviert / deaktiviert Unterstützung des XMS-Host
/LRnnn		Reserviert nnn KB Speicher unter 1 MB für Programme

Weitere Einzelheiten zu der Umgebungsvariablen BLINKER finden Sie im Kapitel 2.

Fehlersuche in Protected Mode-Programmen

Allgemein

Weil DOS Extended Programme im Protected Mode laufen, profitieren sie automatisch von den zusätzlichen Prüfungen, die der Prozessor im Protected Mode durchführt. Fehler wie Segmentüberschreitungen, ungültige Zeiger und dergleichen werden sofort erkannt und erzeugen eine allgemeine Schutzverletzung (GPF)

Mit Ausnahme von CA-Clipper, der seinen eigenen Quellcode Debugger hat (siehe unten), benötigt man zum Debuggen von DOS Extended Programmen Dbugger wie den NuMega SoftICE/W oder den Periscope für Windows. Diese beiden Produkte können auch zur Fehlersuche in Windows und Real Mode DOS Programmen verwendet werden, bieten also eine Schnittstelle für alle drei Umgebungen.

Blinkinc verbessert ständig die Features und Funktionalität von Blinker. Schauen Sie daher bitte in das Dokument DEBUG.WRI im Blinker Verzeichnis für die neuesten Informationen über die Unterstützung von Protected Mode Debuggern.

Wenn Sie keinen Debugger einsetzten, dann wird eine Schutzverletzung, wie z.B. ein ungültiger Zeiger, an die Standard- behandlungsroutine innerhalb des DOS Extenders weitergeleitet. Diese Routine gibt dann alle Register, den gerade ausgeführten Code und die Stackumgebung aus und keht dann zum DOS zurück. Diese Ausgabe wird an die Standardausgabe geschickt. Sie kann bei Bedarf über die Befehlszeile in eine Datei umgeleitet werden, z.B.:

TEST > TEST.GPF

Der unten folgende Abschnitt mit der Überschrift ‘Analyse von Allgemeinen Schutzverletzungen (GPFs)’ enthält eine genaue Beschreibung, wie man die Ausgabe eines GPFs untersucht.

Hinweis zu CA-Clipper 5.x

Der Debugger von CA-Clipper 5.x kann im Protected Mode verwendet werden, wenn man ihn in das Programm linkt. Dies geschieht mit dem Befehl:

```
FILE <path>CLD.LIB
```

am Ende des Linkskripts. Eventuell muß man einen Pfad angeben, wenn sich die Datei nicht im aktuellen Verzeichnis befindet. Zur Laufzeit wird der Debugger gestartet mit der Taste Alt-D oder durch Einfügen des Funktionsaufrufes ALTD() in den Code.

Der Standalone Debugger CLD.EXE ist nicht zu Protected Mode kompatibel und kann daher nicht zur Fehlersuche in Protected Mode Programmen verwendet werden.

Analyse von Allgemeinen Schutzverletzungen (GPFs)

Überblick zu GPFs

Analyse eines Beispiel GPF

Meldung eines GPF an den technischen Support

Besondere GPFs und Ihre üblichen Ursachen

Analyse von Allgemeinen Schutzverletzungen (GPFs)

Überblick zu GPFs

Allgemeine Schutzverletzungen (‘general protection faults’, auch ‘general protection exceptions’, ‘GPEs’, oder ‘general protection violations’) treten auf, wenn der soeben ausgeführte Programmcode versucht, eine ungültige Anweisung auszuführen, auf ein ungültiges Segment oder außerhalb des gültigen Bereichs eines Segments zuzugreifen. Diese Fehler sind nicht Fehler des DOS-Extenders oder des Linkers, sie werden vom Programm im Protected Mode verursacht.

Ein GPF-Dump (Speicher- und Registerauszug) wird auf die Standard-Ausgabe gegeben, kann aber auch durch eine Umleitung auf DOS-Kommandozeilenebene in eine Datei umgeleitet werden:

```
TEST > TEST.GPF
```

Oder, um mehrere GPFs in einer Datei zu sammeln :

```
TEST >> TEST.GPF
```

Analyse von Allgemeinen Schutzverletzungen (GPFs)

Analyse eines Beispiel GPF

In order to correctly analyse a GPF dump it is necessary to create a .MAP file for the program using the MAP S,A link script command. In addition, specifying the X parameter to MAP will allow you to easily locate the OBJ module containing the offending code, although the MAP file is significantly larger so linking may be a little slower.

Beispiel:

```
BLX286: 1313: Ausnahmefehler 0D: Allgemeine Schutzverletzung, code=08DC
Akiver Host ist DPMI (v4.10 i486 5264 Kb)

Reg Wert GrenzeBasis    Flags Segment Modul   Datei
CS  0117 FD8B  808EFC10 FB00    04      T     C:\TEST\T.EXE <-SEGMENT|
DS  0147 FFFF  808CF490 F300    0A      T     C:\TEST\T.EXE  NUMMER |
```

```
ES  00B7 00FF  80900000 F300    **                              (04)   |
SS  0147 FFFF  808CF490 F300    0A       T     C:\TEST\T.EXE          |
                                                                      |
                                                                      |
[Code-Byte CS] C3 BB 0A 00 [IP] 26 C4 1F 26 81 7F 02 55 90 75 05 83       |
                                                                      |
                                                                      |
[Register]  AX=91F7 BX=000A CX=0007 DX=0117  CS:IP=0117:91EE  <----       |
            SI=4988 DI=61DB BP=6192 SP=614E        OFFSET HOLEN    |
            FL=0246 NV UP DI NT NG ZR NA PO NC              (91EE)  |
                                                                      |
                                                                      |
[Stackwert   SS] 08DC 3E13 4988 61DB 61A2 B030 0107 0000 61DB 4988 |
            [SP] 0118 1234 0234 3344 2340 0382 4320 6124 0107 3882 |
                                                                      |
                                                                      |
[Stackrahmen SS] 4988 61DB 61A2 B030 0107 0000 61DB 4988 61DB 0083 |
            [BP] 0118 1234 0234 3344 2340 0382 4320 6124 0107 3882 |
                                                                      |
                                                                      |
Ausschnitt aus der MAP-Datei:                                         |
                                                                      |
 Start     Länge      Name                    Klasse                  |
   .         .          .                       .                    |
   .         .          .                       .                    |
 0004:85C0 00BEEH     DBGAPI_TEXT             CODE                    |
 0004:91AE 00076H     DEBUG_INIT_TEXT         CODE    <--------------      |
 0004:9224 0024AH     DBGSHADO_TEXT           CODE                    |
                                                                      |
                                                                      |
 Addresse        Publics nach Wert                                    |
    .               .                                                 |
 0004:9173  Res  __DBGABORT                                           |
 0004:91E0  Res  DBGINIT               <-----------------------------------
 0004:9224  Res  DISPOUTAT
```

Im oben angeführten Beispiel sehen wir, daß die Schutzverletzung im Modul T innerhalb T.EXE auftrat. Der verursachende Code sitzt in Segment 04. Der Wert des Befehlszeigers (IP) zeigt auf den Offset innerhalb dieses Segments, in diesem Falle 91EE.

Wenn wir nun die MAP-Datei nach dem Offset im Segment 04 durchsuchen, sehen wir, daß der Offset 91EE auf Code im Segment DEBUG_INIT_TEXT verweist. Wurde beim Linklauf die Map-Option ‘A’ angegeben, dann steht eine Auflistung aller Public-Symbole zur Verfügung und wir sehen hier, der Offset 91EE zeigt ein paar Bytes hinter das Public-Symbol DBGINIT.

Das Dienstprogramm BLIGPF.EXE im Blinkerverzeichnis unterstützt diesen Vorgang automatisch. Wenn Sie BLIGPF aufrufen, während der GPF-Auszug noch auf dem Bildschirm ist, zeigt es den Namen des verursachenden Public-Symbols an.Wenn der GPF Dump in eine Datei umgeleitet wurde, zeigen Sie diese Datei auf dem Bildschirm an, z.B. mit TYPE TEST.GPF auf dem DOS-Prompt, und starten dann BLIGPF.EXE.

Um herauszufinden, ob sich die Funktion DBGINIT nun im Programmquellcode oder in einer externen Bibliothek befindet, und wie der genaue Modulname lautet, verwenden Sie den MAP Befehl mit dem Parameter X, um eine Darstellung aller Segmente mit allen zugehörigen Modulen zu erhalten.

```
 Detailed segment map

 Start    Length Name             Class  Module    ACBP
 .        .      .                .      .         .
 0004:85C0   00BEEH DBGAPI_TEXT      CODE   DBGAPI    48
 0004:91AE   00076H DEBUG_INIT_TEXT  CODE   DBGINIT   28
```

```
 0004:9224    0024AH DBGSHADO_TEXT    CODE   DBGSHADO  48
 .        .      .                .      .      .
```

Hier sieht man, daß das Modul mit dem fehlerhaften Code DBGINIT heißt. Falls es noch nicht sofort ersichtlich ist, erhält man die genaue .OBJ oder .LIB Datei, in dem das Modul enthalten ist, nach Einfügen des VERBOSE Befehls ins Linkskript durch Betrachtung der Ausgabe von Blinker.

Die Zeile beginnend mit '[Code byte CS]' in der GPF Ausgabe zeigt links von '[IP]' die Code Bytes, die zuletzt ausgeführt wurden. Rechts von '[IP]' stehen die Bytes, die noch ausgeführt werden sollen und den GPF verursacht haben. Jeder, der sich in Assembler auskennt, kann sich mit dem DOS Hilfsprogramm DEBUG den Assemblercode anzeigen lassen für die Anweisung, die den GPF erzeugt. Um z.B. beim obigen GPF den DEBUG aufzurufen, die Code Bytes hinter '[IP]' einzugeben und die fehlerhafte Anweisung zu disassemblieren, müßten Sie folgendes eintippen:

```
C:>DEBUG

-E100 26 C4 1F 26 81 7F 02 55 90 75 05 83

-U100
nnnn:0100 26              ES:
nnnn:0101 C4 1F           LES    BX,[BX]
nnnn:0103 26              ES:
nnnn:0104 81 7F 02 55 90  CMP    WORD PTR [BX+02],9055
. . .

-Q
C:>
```

Die disassemblierten Befehle zeigen, daß der schuldige Befehl
LES ES:BX,[BX] war, was normalerweise ausreicht, um die Ursache zu finden und zu beseitigen.

Die Anzeige von Sternen (****) in den Segmentregistern zeigt an, daß ein Selektor Bestandteil eines Overlays ist und nicht zu dieser Applikation gehört. Sterne im Code Byte oder im Stack zeigen Daten außerhalb einer Segmentgrenze an.

Tritt in einem Dual Mode-Programm ein GPF auf, kann dieser in einem überlagerten Segment verursacht worden sein. In diesem Fall ist der Wert des CS relativ hoch und es wird keine Segmentnummer angezeigt. Linken Sie in diesem Fall das Programm als ein Extended Mode-Programm, wodurch die Overlays entfernt werden und diese Segment im Map-Listing aufgeführt werden. Nun können Sie anhand des Dumps den fehlerverursachenden Code lokalisieren. Alternativ können Sie die Code-Segmentgrenze aus dem Dump verwenden um in der Map-Datei das nächstliegende Overlay zu suchen.

Analyse von Allgemeinen Schutzverletzungen (GPFs)

Meldung eines GPF an den technischen Support

Wenn Sie einen GPF Dump an den technischen Support melden wollen, fügen Sie bitte folgende Informationen hinzu, ohne die ein Support nicht möglich ist.:

- Den GPF Dump (sehen Sie in den vorherigen Abschnitten, wie man ihn in eine Datei speichert).
- Details zur Umgebung, in der das Programm läuft, z.B. eine DOS-Box von Windows 95 OEM 2 auf einem Pentium 200MHz MICRON Millenia.
- Die Teile der 'Publics by Value' und 'Detailed segment map' Abschnitte der MAP Datei, in denen

der GPF auftritt, wie in den vorherigen Abschnitten beschrieben.

- Eine Beschreibung, was das Programm tut oder versucht zu tun, wenn der Fehler auftritt, und an welcher Stelle im Programm. Dazu gehören auch die Quellzeilen in der Umgebung des Fehlers.
- Ein Hinweis, ob der Fehler jedesmal auftritt, wenn die Stelle im Programm erreicht wird, oder ob er nur sporadisch vorkommt.
- Die Ergebnisse von Linken und Ausführung eines kleinen Testprogramms (z.B. print "Hello world") in derselben Umgebung.
- Die Ergebnisse von Linken und Ausführung eines kleinen Testprogramms mit den Quellzeilen , die das Problem verusachen, in derselben Umgebung.

Analyse von Allgemeinen Schutzverletzungen (GPFs)

Besondere GPFs und Ihre üblichen Ursachen

- Ein GPF in der Funktion __FPINSTALL87 oder __FARMALLOC weist darauf hin, daß die entsprechende Bibliothek BLXxxxnn.LIB nicht mit SEARCH gelinkt wurde.
- Ein GPF in der Funktion __FREEF oder __BEEP_ in einem Clipper 5.x Programm weist darauf hin, daß die entsprechende Bibliothek BLXCLPnn.LIB nicht mit SEARCH gelinkt wurde.
- Ein GPF in der Funktion __AHINCR weist daraus hin, daß die entsprechende Datei C0LPnn.OBJ nicht als erste Objektdatei im Linkskript angegeben wurde.
- Ein GPF im ALLOC Modul eines Clip4Win Programms
  weist darauf hin, daß die Bibliothek CLIP4WIN.LIB nicht mit SEARCH gelinkt wurde.
- Ein GPF in der Funktion _MALLOC weist darauf hin, daß eventuell ein C Modul nicht im LARGE oder im HUGE Speichermodell kompiliert wurde.

Transparente Interrupt-Unterstützung

BIOS-Interrupts

DOS-Interrupts

Transparente Interrupt-Unterstützung

BIOS-Interrupts

Folgende BIOS-Interruptaufrufe werden transparent behandelt:

INT 10h	Video	Funktionen 00h - 1Ch
INT 11h	Equipment-Liste	
INT 12h	Speichergröße ausgeben	
INT 13h	Laufwerke	Funktionen 00h - 19h
INT 14h	RS232	Funktionen 00h - 05h
INT 16h	Tastatur	Funktionen 00h – 12h
INT 17h	Drucker	Funktionen 00h - 02h
INT 1Ah	Uhr	
INT 1Bh	Ctrl Break-Taste	
INT 1Ch	Timer Tick	
INT 33h	Maus	
INT 7Ah	IPX / SPX	

Transparente Interrupt-Unterstützung

DOS-Interrupts

Folgende DOS-Interruptaufrufe werden transparent behandelt:

INT 21h Funktionen 00h - 6ch außer:

0FH	Datei öffnen (fcb)
10H	Datei schließen (fcb)
14H	Sequentielles Lesen von Dateien
15H	Sequentielles Schreiben von Dateien
16H	Dateien erzeugen
21H	Wahlfreies Lesen von Dateien
22H	Wahlfreies Schreiben von Dateien
23H	Ermitteln der Dateigröße
24H	Setzen der relativen Datensatzgröße
27H	Wahlfreies Lesen von Datenblöcken
28H	Wahlfreies Schreiben von Datenblöcken
31H	TSR
5DH	Verschiedene DOS interne Funktionen
5EH	Microsoft Network-Funktionen 3 und höher
5FH	Netzwerk-Funktionen 6 und höher
65H	Länderinformation abfragen

INT 23h	Ctrl C-Taste
INT 25h	Absolutes DOS-Lesen
INT 26h	Absolutes DOS-Schreiben

DPMI API-Funktionsunterstützung

Möchten Sie direkt für das DPMI progammieren, dann stehen folgende Aufrufe zur Verfügung, die von Blinkers DOS-Extender - unabhängig davon, welcher Host installiert ist - unterstützt werden:

0000h LDT Deskriptoren allozieren
0001h LDT Deskriptoren freigeben
0002h Real Segment in Selektor
0003h Selektor-Increment abfragen
0004h Selector LOCK
0005h Selector UNLOCK
0006h Segment-Basisadresse abfragen
0007h Segment-Basisadresse setzen
0008h Segmentgrenze setzen
0009h Deskriptor-Zugriffsrecht setzen
000Ah Data-Alias für Code-Selector setzen
000Bh Deskriptor abfragen
000Ch Deskriptor setzen
0100h DOS Speicherblock allozieren (max. 64 KB)
0101h DOS Speicherblock freigeben
0102h DOS Speicherblock Größe neu bestimmen (max. 64 KB)
0200h Real Mode-Interrupt-Vektor abfragen
0201h Real Mode-Interrupt-Vektor setzen
0202h Prozessor-Exception-Vektor abfragen
0203h Prozessor-Exception- Vektor setzen
0204h Protected Mode-Interrupt-Vektor abfragen
0205h Protected Mode-Interrupt-Vektor setzen
0300h Real Mode-Interrupt simulieren
0301h Real Mode-Prozedur mit RETF frame aufrufen
0303h Real Mode-call-back-Adresse zuweisen
0304h Real Mode-call-back-Adresse freigeben
0400h Version abfragen
0500h Information über freien Speicher abfragen
0600h-604h Führe nichts aus, Rücksprung mit gelöschten Carry-Flags

Windows API-Funktionsunterstützung

Die folgenden Windows API-Funktionen werden durch den Blinker DOS-Extender transparent unterstützt:

DOS3CALL
ENUMTASKWINDOWS
FATALAPPEXIT
FATALEXIT
FREELIBRARY
GETCURRENTTASK
GETDOSENVIRONMENT
GETMODULEHANDLE

GETMODULEUSAGE
GETPROCADDRESS
GETVERSION
GETWINFLAGS
GLOBALALLOC
GLOBALFREE
GLOBALHANDLE
GLOBALLOCK
GLOBALREALLOC
GLOBALSIZE
GLOBALUNLOCK
INITTASK
LOADLIBRARY
LOCALINIT
LOCKSEGMENT
MESSAGEBOX
UNLOCKSEGMENT
__AHINCR
__AHSHIFT
__WINFLAGS

Windows-Programme

Dynamisches Linken und DLLs

Moduldefinitionsdateien

Der BLILIB Import-Bibliotheksmanager

Linken von 16-Bit-Windows-Programmen

Linken von 32-Bit-Windows-Programmen

Moduldefinitionen - eine Quick-Referenz

Moduldefinitionsbefehle

Dynamisches Linken und DLLs

Dynamisches Linken gg. Statisches Linken

Exporte und Importe

Vorteile der Verwendung von DLLs

Kodierung für DLLs

Dynamisches Laden von DLLs zur Laufzeit

Dynamisches Linken und DLLs

Dynamisches Linken gg. Statisches Linken

In höheren Programmiersprachen wie C, PASCAL, oder FORTRAN, muß der Quelltext einer Anwendung compiliert und mit verschiedenen Bibliotheken gelinkt werden, um ein ausführbares Programm zu erzeugen. Werden diese Bibliotheksfunktionen in das Programm gelinkt, sind sie ein fester Bestandteil dieses Programms. Alle Aufrufe von Bibliotheksfunktionen werden zur Linkzeit erfüllt, daher spricht man hier von ‘Statischem Linken’.

Dynamisches Linken unterstützt einen Mechanismus, bei dem Anwendungen mit Bibliotheken erst zur Laufzeit verbunden werden. Diese Bibliotheken sind eigene Programme und nicht Bestandteil des ausführbaren Programmes, wie beim Statischen Linken. Solche Dateien nennt man ‘Dynamische Linkbibliotheken’ (dynamic link libraries, DLLs) um zu betonen, daß sie zu einer Anwendung beim Programmstart und -ablauf dazugelinkt werden und nicht bei dessen Erzeugung.

DLLs differieren erheblich von statischen Linkbibliotheken. Grundlegende Unterschiede sind:

- Statische Linkbibliotheken bestehen aus LIB-Dateien, die im Grunde eine Ansammlung von Objektdateien sind, während dynamische Linkbibliotheken sich in separaten ausführbaren Dateien befinden, die von Windows bei Bedarf in den Hauptspeicher geladen werden.
- Statische Linkbibliotheken enthalten nur Code und Daten, da diese als Objektdateien in die Bibliothek abgelegt wurden. DLLs unter Windows wiederum können Code, Daten, aber auch Ressourcen, wie etwa Bitmaps, Icons oder Cursordefinitionen, enthalten, da diese als ausführbare Programmteile Bestandteil der Bibliothek sind.
- Statische Linkbibliotheken verwenden immer das Datensegment der Anwendung, während DLLs ihr eigenes Datensegment besitzen können (dies ist meistens der Fall).

Bei Zugriff eines Programmes auf eine DLL lädt das Betriebssystem die DLL in den Speicher, löst Referenzen auf Funktionen der DLL auf (d.h. es ermöglicht so der Anwendung den Aufruf dieser Funktionen) und entlädt die DLL wieder, sobald sie nicht mehr benötigt wird, aus dem Speicher.

Beachten Sie, daß, obwohl Anwendungen Funktionen in einer DLL verwenden können - auch eine DLL kann Funktionen in einer anderen DLL aufrufen - eine DLL jedoch niemals auf Funktionen des aufrufenden Programmes zugreifen kann. Das liegt daran, daß der Importmechanismus eben nur einen Import von bezeichneten Modulen unterstützt.

Dynamisches Linken kann auf zwei Arten erfolgen:

- Implizites dynamisches Linken wird zur Ladezeit vom Betriebssystem durchgeführt. Lädt das Betriebssystem eine Anwendung oder eine DLL, werden auch alle verwendeten DLLs, d.h. jede DLL, die von der Anwendung und jede DLL, die wieder von einer der geladenen DLLs benötigt wird, vorher geladen.
- Explizites dynamisches Linken wird vom Programm selbst oder einer DLL zur Laufzeit ausgeführt. Durch Betriebssystemaufrufe können DLLs geladen und Symbole innerhalb dieser DLLs aufgelöst werden.

Dynamisches Linken und DLLs

Exporte und Importe

Um nun eine Funktion anderen .EXE- oder .DLL-Dateien zugänglich zu machen - zusammengefaßt nennt man sie auch Module - exportiert nun die DLL die im EXPORTS-Abschnitt der Moduldefinitionsdatei bestimmte Funktion. Dies kann auf zwei Arten geschehen: entweder anhand des Namens oder aber anhand der Funktionsnummer, auch Ordnungsnummer genannt. Beide Alternativen werden im Abschnitt ‘Kodierung für DLLs’, weiter hinten in diesem Kapitel, besprochen.

Möchte ein Modul eine Funktion, die Bestandteil eines anderen Moduls ist, verwenden, importiert es diese Funktion, um ein dynamisches Link auf den Code dieser Funktion zu erzeugen, und zwar über den Namen oder über die Ordnungsnummer. Zwei Möglichkeiten zum Import einer Funktion in ein Modul stehen zur Verfügung:

- Das Modul wird mit einer Importbibliothek gelinkt, die Information über diese Funktion enthält.
- Die einzelne Funktion wird im IMPORTS-Abschnitt der Moduldefinitionsdatei angegeben.

Sollen z.B. Programme für Windows mit den normalen Laufzeitbibliotheken der Programmiersprache gelinkt werden, ist es nötig, auch die Bibliothek LIBW.LIB (MS Visuual C++) oder IMPORTS.LIB (Borland C++) einzubinden, in denen die Definitionen aller Windows API-Funktionen abgelegt sind. Hierbei handelt es sich um sogenannte Importbibliotheken, die dem Linker die Auflösung der Referenzen (Aufrufe) auf Funktionen, welche sich in den DLLs befinden, ermöglicht. In diesem Fall handelt es sich um Referenzen auf Funktionen des Windows Applikationsprogrammierungs-Interface (API) in den System-DLLs, wie etwa USER, KERNEL und GDI.

Sobald ein Programm auf eine Funktion verweist, die in der Importbibliothek festgelegt wurde, legt der Linker den Namen der Bibliothek wie auch die Referenz auf die einzelne Funktion im Header der .EXE-Datei ab. Beim Laden des Programmes werden diese Referenzen mit den entsprechenden Funktionen der dazugehörigen DLLs aufgelöst.

Es werden nur Funktionen aus der DLL exportiert, die explizit von anderen Modulen aufgerufen werden. Funktionen, die eine DLL intern verwendet, müssen nicht exportiert werden.

Die Erzeugung einer Importbibliothek wird mit einem Import-Bibliotheksmanager durchgeführt. Für den Blinker wird hierzu das Programm BLILIB.EXE mitgeliefert, bei Microsoft Visual C++ liegt das Programm IMPLIB.EXE bei. Weitere Einzelheiten zur Verwendung des Bibliothekmanagers BLILIB.EXE finden Sie weiter unten in diesem Kapitel im Abschnitt ‘Der BLILIB Import-Bibliotheksmanager’.

Dynamisches Linken und DLLs

Vorteile der Verwendung von DLLs

Die Verwendung von DLLs gegenüber statischen Linkbibliotheken bietet mehrere Vorteile: Zum einen ist der Programmcode allen Applikationen gleichzeitig zugänglich, zum anderen bedeutet auch das Erneuern oder Ersetzen von Teilen eines Programmsystems eine wesentliche Vereinfachung - selbst wenn große Systeme sich in diesen Modulen befinden, ist die Verwaltung bedeutend leichter.

Da Programmcode, der von mehreren Anwendungen verwendet wird und sich in einer DLL befindet, nur noch einmal in der DLL und nicht in jedem Programm gespeichert ist, verringert sich die Größe der .EXE-Dateien und demzufolge der benötigte Speicherplatz auf der Festplatte deutlich.

Auch das Ändern, Erweitern und das Update einer DLL kann unabhängig von den Programmen erfolgen, d.h. alle Programme, die mit dieser DLL arbeiten, müssen nicht neu compiliert und gelinkt werden. Man könnte zum Beispiel Informationen und Programmteile in verschiedenen Sprachen in unterschiedlichen DLLs liefern, wobei dem Anwender bei der Installation die entsprechende DLL kopiert oder umbenannt wird.

Nehmen wir an, viele verschiedene Programme arbeiten auf einem Rechner zusammen und greifen gleichzeitig auf gemeinsame DLLs zu. Wird nun die DLL durch eine neuere Version ersetzt, in der vielleicht schnellere Algorithmen integriert, Bug Fixes durchgeführt wurden etc., profitieren alle Anwendungen, die mit dieser DLL arbeiten, gleichzeitig davon.

Das Update einer DLL bedarf, wenn Anwendungen ohne Update weiter mit ihr betrieben werden sollen, einiger Regeln. Es dürfen sich die Namen der exportierten Funktionen nicht ändern (falls sie über den Namen exportiert werden), auch Ordnungsnummern, Aufrufkonventionen und Parameterlisten müssen gleich bleiben. Bei Einhaltung dieses Kompatibilitätslevels sind Updates der .EXE-Dateien aller zugreifenden Anwendungen nicht mehr nötig.

Code und Daten, wie auch Einzelheiten zu deren Implementierung, bleiben Anwendungen verborgen, wenn sie in DLLs abgelegt sind. Eine DLL bietet folglich eine klar definierte Schnittstelle zu einem abstrakten Datentyp (ADT), der von Programmen zwar benutzt werden kann, Informationen über dessen Implementation ihnen jedoch nicht zugänglich sind. Änderungen im internen Ablauf oder der internen Datenstrukturen können vorgenommen werden, ohne die Programme zu modifizieren. Ein Beispiel wäre ein austauschbarer Datenbanktreiber mit einer Schnittstelle für höhere Zugriffsfunktionen.

Nachteile von DLLs sind jedoch in dem größeren Entwicklungsaufwand und der höheren Komplexität bei der Verwaltung mehrerer Anwendungen auf der Platte zu finden.

DLLs sind keine Tasks. Sie besitzen keine eigenen Stacks, sondern verwenden den des aufrufenden Programms. Dies bringt oft Verwirrung, da Programmierer stets daran denken müssen, daß bei DLLs, die ihre eigene DGROUP besitzen, das Stacksegment nicht gleich dem Datensegment ist. In diesem Fall muß mit Zeigern sehr sorgfältig umgegangen werden, damit sichergestellt ist, daß sie auf die richtige Adresse im richtigen Segment verweisen. Lesen Sie hierzu den nächsten Abschnitt ‘Kodierung für DLLs’.

Dynamisches Linken und DLLs

Kodierung für DLLs

Eine Funktion, die von anderen Modulen aufgerufen wird, muß als FAR-Typ deklariert und im EXPORTS-Abschnitt der Moduldefinitionsdatei dieser Bibliothek aufgeführt sein. Mehr dazu im Abschnitt ‘Moduldefinitionsdateien’ weiter unten in diesem Kapitel.

Werden DLL-Funktionen in niedrigeren Programmiersprachen wie Assembler, C, C++ und PASCAL erstellt, müssen gewöhnlich extra Compilerschalter angegeben werden, um den richtigen Einstiegs- und Ausstiegscode der exportierten Funktionen zu erzeugen, es sei denn, sie wurden explizit als Exportfunktionen bereits im Quelltext deklariert (hierzu steht gewöhnlich das Schlüsselwort __export oder ein Äquivalent zur Verfügung). Informationen zur Erstellung von DLL-Code in einer dieser Sprachen finden Sie in Ihrem Compilerhandbuch.

An DLL-Funktionen, die in CA-Clipper entwickelt wurden, werden keine speziellen Anforderungen gestellt, denn Blinker verwaltet sowohl Einsprung als auch Aussprung der exportierten Funktion transparent.

Wie bereits vorher angesprochen, gibt es zwei Wege, Funktionen aus einer DLL zu exportieren: über den Namen oder anhand der Ordnungsnummer, je nachdem wie sie im EXPORTS-Abschnitt der Moduldefinitionsdatei festgelegt wurden. Eine oder auch beide Methoden können für jede Funktion einsetzt werden, auch eine Kombination beider Methoden in der gleichen DLL ist möglich.

Export über den Namen

Soll eine Funktion über ihren Namen exportiert werden, so gibt man ihr typischerweise den Namen, unter dem sie der DLL intern bekannt ist. Aber ihr kann auch ein anderer Name zugewiesen werden, indem dieser Name in der EXPORT-Definition angegeben wird. Bei Einsatz von C++ oder anderen Programmiersprachen, die Funktionsnamen verändern (abkürzen oder mit einem ‘_’ ergänzen) muß der verkürzte oder ergänzte Name angegeben werden.

Beispiel:

```
EXPORTS myfunction
```

exportiert die Funktion ‘myfunction’ unter ihrem eigenen Namen, während

```
EXPORTS neuername=myfunction
```

verwendet den Namen ‘neuername’ beim Export der Funktion ‘myfunction ‘.

Der exportierte Name ist in einer Tabelle im Header der exportierenden DLL gespeichert, zusammen mit der Adresse der Funktion innerhalb der DLL . Der Name der DLL und der exportierte Funktionsname wird auch im Header eines jeden Moduls, das auf diese Funktion verweist, gespeichert. Beim Laden des Moduls sucht das Betriebssystem den Funktionsnamen im Header der DLL und extrahiert die Adresse der Funktion aus der Tabelle.

Trotz eines geringen Mehraufwandes zur Ladezeit, wenn alle Namen gesucht werden, ist diese Methode für eine DLL doch sehr sinnvoll, da Funktionen neu geschrieben oder innerhalb einer DLL verlagert werden können, ohne das aufrufende Modul in irgendeiner Weise verändern zu müssen.

Export über die Ordnungsnummer

Wird eine Funktion anhand einer Ordnungsnummer exportiert, wird ihr eine einmalige Ordnungsnummer

innerhalb der DLL zugewiesen, die in der EXPORT-Definition aufgeführt ist. Mit dem Schlüsselbegriff NONAME kann ein Funktionsaufruf über den Namen verhindert werden.

Beispiel:

```
EXPORTS myfunction @ 1 NONAME
```

exportiert die Funktion ‘myfunction’ nur mit der Ordnungsnummer 1, während:

```
EXPORTS neuername=myfunction @1
```

sowohl den Namen ‘neuername’ als auch die Ordnungsnummer 1 zum Export der Funktion ‘myfunction’ verwendet.

Beim Export einer Funktion über eine Ordnungsnummer wird die Adresse innerhalb der DLL in einer Tabelle in der Reihenfolge der Ordnungsnummern gespeichert, was zur Laufzeit ein schnelleres Auffinden ermöglicht. Der Name der DLL und die Ordnungsnummer werden im Header jedes Moduls, das auf diese Funktion verweist, abgelegt. Soll das Modul geladen werden, sucht das Betriebssystem anhand der Ordnungsnummer im Header der entsprechenden DLL nach der Adresse der angeforderten Funktion.

Diese Methode bedeutet weniger Aufwand zur Laufzeit, sie ist jedoch etwas unflexibler. Funktionen müssen ihre Ordnungsnummer in der DLL beibehalten, sollen die aufrufenden Module nicht geändert werden. Ein Vorteil des Imports über Ordnungsnummer ist jedoch seine höhere Sicherheit: da keine Funktionsnamen in der DLL abgelegt sind, stehen sie auch demjenigen, der unerlaubt auf die DLL zugreifen möchte, nicht zur Verfügung.

DLLs und der Stack

DLLs besitzen – anders als Task Module - keinen eigenen Stack. Statt dessen verwenden sie das Stack-Segment der aufrufenden Task. Dies kann nun zu Problemen führen, wenn ein Modul eine Funktion in der Annahme aufruft, die Register DS und SS enthielten die gleiche Adresse. Das Problem tritt meistens in Small-Model- oder Medium-Model-DLLs auf. Zeiger dieser Modelle sind vom Typ ‘Near’. So betrachten etwa viele C-Funktionen aus der Laufzeitbibliothek die Segmente DS und SS als gleich, sind also nicht geeignet, in einer DLL abgelegt zu werden.

Gehen Sie daher vorsichtig damit um, existierenden Code, wie etwa Compilerbibliotheken oder Zusatzbibliotheken, in eine DLL zu legen, wenn Ihnen der Quelltext nicht zugänglich ist.

Dies ist ein weiterer Grund, warum, wie im vorhergehenden Abschnitt erwähnt, Low-Level-Code mit den richtigen Compiler-Optionen übersetzt werden muß. CA-Clipper ist die bekannte Ausnahme dieser Regel. Sein Code wird in Pseudo-Code übersetzt, der bei Segmentregister nichts voraussetzt.

Tritt ein Problem mit einer Funktion in einer DLL auf, testen Sie denselben Code im .EXE-Hauptprogramm, um sicher zu sein, daß dieses Problem nicht auf der Annahme beruht, DS und SS seien gleich.

Versionskontrolle

Der hauptsächliche Vorteil von DLLs liegt darin, daß man sie transparent ersetzen kann, ohne den Rest des Programmes verändern zu müssen. Dies kann jedoch auch zu Problemen führen, wenn man eine DLL durch eine neuere oder eine frühere Version ersetzt, die zwar bezüglich Funktionsnamen oder Ordnungsnummern kompatibel zu sein scheint, es in der Tat jedoch nicht ist.

Aus diesem Grund ist es dringend erforderlich, daß der Programmierer eine Art Versionskontrolle oder eine Funktionsüberprüfung für alle DLLs implementiert, die sich im Laufe der Zeit signifikant ändern.

Im einfachsten Fall könnte dies eine einzelne Funktion sein, die eine Versionsnummer zurückliefert, mit

der auf eine bestimmte DLL-Version getestet werden kann. Besser wäre die Rückgabe eines Informationssatzes, der anzeigt, ob bestimmte Funktionen verfügbar sind oder nicht.

Dynamisches Linken und DLLs

Dynamisches Laden von DLLs zur Laufzeit

Der vorhergehende Abschnitt diskutiert die Implementierung von DLLs, die statisch zum .EXE-Hauptprogramm gelinkt werden. Deshalb müssen Modul- und Funktionsnamen der DLLs und Einzelheiten über ihre Funktionalität zur Kompilier- und Linkzeit zur Verfügung stehen.

Eine ungleich leistungsstärkere Besonderheit von DLLs ist die Fähigkeit des dynamischen Ladens und Ausführens neuer DLLs, fast ohne Wissen um Namen und ihrer Funktionalität. Dadurch können Programme geschrieben werden, die in einem späteren Stadium in völlig neuer Funktionalität erweitert werden, wie z.B. durch Implementierung von Datenbanktreibern oder ganz neuen Modulen wie Reportgeneratoren oder einer Backup-Option.

Das Windows 3.x-API bietet drei Funktionen, die zum dynamischen Laden von DLLs verwendet werden können: LoadLibrary(), GetProcAddress() und FreeLibrary().

LoadLibrary() nimmt als Parameter den Namen der zu ladenden DLL, und wenn die Bibliothek erfolgreich geladen wurde, liefert die Funktion eine einmalige Nummer, einen Handle, zurück, um die Bibliothek für zukünftige Operationen zu identifizieren. Schlägt der Ladevorgang fehl, enthält der Rückgabewert den in den Standard-Dateien WINDOWS.H oder WINDOWS.INC festgehaltenen Fehlercode, und ist immer kleiner als der HINSTANCE_ERROR-Wert 32, der ebenfalls in WINDOWS.H definiert ist. Wird ein Verzeichnispfad mit dem DLL-Namen angegeben, sucht Windows nur in diesem Verzeichnis, bei Angabe nur des Namens werden jedoch nacheinander durchsucht:

- Das Verzeichnis, von dem aus das .EXE-Hauptprogramm geladen wurde (der sog. Ladepfad).
- Das aktuelle Verzeichnis.
- Die über die Umgebungsvariable PATH angegebenen Pfade.

Um die Adresse einer exportierten Funktion in der soeben geladenen DLL zu erhalten, wird der Handle, der von LoadLibrary() zurückgeliefert wurde, und ein Zeiger auf den Namen der gewünschten Funktion an die Funktion GetProcAddress() übergeben. Wenn eine Funktion mit diesem Namen in der DLL existiert, dann wird ein Zeiger auf den Funktionscode, andernfalls ein Nullzeiger zurückgeliefert, der anzeigt, daß eine Funktion dieses Namens nicht in der DLL existiert oder die DLL Funktionen nicht über den Namen exportiert.

Schließlich, vor Verlassen des Programmes, sollte die Funktion FreeLibrary() mit dem Handle der Bibliothek aufgerufen werden, um von der DLL belegten Speicher und Ressourcen wieder freizugeben. Die Beispielsprogramme im Unterverzeichnis \C\DLL\DYN des Blinker-Installationsverzeichnisses zeigen, wie ein Windows-Programm und ein DOS-Extender-Programm beide dynamisch die selbe Windows-DLL laden und ausführen. Der relevante Teil des Quelltextes eines Windowsprogramms sieht wie folgt aus:

```
if ( (hLibInst = LoadLibrary (LibName)) > HINSTANCE_ERROR )
    {
    TestPtr = (TEST) GetProcAddress (hLibInst, FunName);
    if ( TestPtr )
        {
```

```
        wsprintf (msgbuf, "%s zurückgegeben ", FunName);
        strcat (msgbuf, (*TestPtr) (1));
        WasOK++;
        }
    else
        wsprintf (msgbuf, "%s%s%s", FunName,
                          " existiert nicht in Bibliothek ",
                          LibName);
    }
else
    {
    if (hLibInst = 2)
        strcpy (msgbuf, "Nicht gefunden ");
    else
        strcpy (msgbuf, "Fehler beim Laden von ");
    strcat (msgbuf, LibName);
    }

/* Ergebnis anzeigen */
MessageBox (hwnd, msgbuf, "",
    MB_APPLMODAL | (WasOK ? MB_ICONINFORMATION :
    MB_ICONEXCLAMATION) | MB_OK);

/* Bibliothek freigeben */
    if (hLibInst > HINSTANCE_ERROR)
        FreeLibrary (hLibInst);
```

Der Blinker DOS-Extender bietet die gleichen drei Funktionen wie das Windows-API für Programmiersprachen auf niedrigerem Level wie C, PASCAL und Assembler, deshalb sieht ein Programm für den DOS-Extender, das eine DLL dynamisch lädt und ausführt, fast identisch aus und verhält sich auch in gleicher Weise. Sehen Sie die Beispiele in den \C\DLL Unterverzeichnissen für weitere Informationen, oder die Windows-SDK-Dokumentation über Funktionen zur Bearbeitung von DLLs.

Äquivalente Funktionen für Hochsprachen werden vom DOS-Extender ebenso unterstützt, sie sind im Abschnitt 'Spezielle Hinweise zum DLL-Code' in Kapitel 3 beschrieben.

Moduldefinitionsdateien

Überblick über Moduldefinitionsdateien

Moduldefinitionsdatei-Befehlssyntax

Beispiel einer Moduldefinitionsdatei

Moduldefinitionsdateien

Überblick über Moduldefinitionsdateien

Moduldefinitionsdatei-Befehlssyntax

Beispiel einer Moduldefinitionsdatei

Neben der Freeformat-Link-Skriptdatei, in der die einzubindenden .OBJ- und .LIB-Dateien angegeben werden, verwendet Blinker eine Moduldefinitionsdatei zur Angabe zusätzlicher Informationen eines ausführbaren Windows-Programms. Da das Windows-.EXE-Format ebenso für DOS-Extender-Programme verwendet wird, können viele Moduldefinitionsbefehle auch für DOS-Extender-Programme genutzt werden.

Format und Befehle einer Moduldefinitionsdatei entsprechen, mit ein oder zwei Zusätzen zur Erweiterung der Funktionalität, denen, die für Microsoft LINK, TLINK etc. eingesetzt werden.

Blinker unterstützt Moduldefinitionsdateien über:

```
DEFFILE <dateiname>
```

Standarddateierweiterung ist hierbei .DEF.

Der Befehl DEFFILE kann in der Link-Skriptdatei an jeder Stelle eingesetzt werden. Es wird jedoch empfohlen, ihn am Ende der Datei zu plazieren, um abzusichern, daß darin angegebene Optionen den Vorrang vor eventuell in der Link-Skriptdatei angegebenen Optionen bekommen.

Alternativ können .DEF-Dateibefehle zwischen den Befehlen DEFBEGIN und DEFEND direkt in der Linkdatei (.LNK) eingefügt werden, um zu vermeiden, daß zwei separate Dateien gepflegt werden müssen:

```
DEFBEGIN                // Beginn der .DEF-Befehle
   NAME   Test
   ...
DEFEND                  // Ende der .DEF-Befehle
```

Diese können wiederum überall in der Linkdatei plaziert werden. Es wird jedoch empfohlen, sie an das Ende der Datei zu setzen.

Blinker sucht in folgender Reihenfolge nach Moduldefinitionsdateien:

- in dem mit den Namen der Datei angegebenen Verzeichnis
- im aktuellen Verzeichnis
- in den über die Umgebungsvariable LIB angegebenen Verzeichnissen

Über dieses Handbuch hinausgehende Informationen bzgl. Moduldefinitionsdateien finden Sie in Ihrer Compilerdokumentation.

Moduldefinitionsdateien

Moduldefinitionsdatei-Befehlssyntax

- Die Groß-/Kleinschreibung wird bei Schlüsselwörtern und vom Benutzer angegebenen Bezeichnern ignoriert, es sei denn, der Befehl MIXCASE EXPORT wird eingesetzt.
- Verwenden Sie ein oder mehr Leerzeichen, Tabulatoren oder den Zeilenvorschub, um ein Anweisungsschlüsselwort von seinen Parametern, oder mehrere Anweisungen voneinander zu trennen. Doppelpunkt (:) oder Gleichheitszeichen (=), die einen Parameter anzeigen, können zwischen einem oder mehreren Leerzeichen, Tabulatoren oder Zeilenvorschüben stehen.
- Eine NAME- oder LIBRARY- Anweisung muß, falls vorhanden, vor allen anderen Anweisungen aufgeführt werden.
- Wenn sich eine Anweisung mit einer anderen Spezifikation in der Moduldefinitionsdatei wiederholt, dann wird die erste Anweisung durch die zweite überschrieben.
- Numerische Argumente werden dezimal angegeben oder durch die Kennzeichnung ‘0x’ hexadezimal.
- Kommentare werden mit einem Semikolon (;) am Beginn oder mitten in einer Zeile gekennzeichnet. Der Rest der Zeile wird dann ignoriert.
- Entspricht ein Zeichenargument einem reservierten Wort, dann muß es in Anführungszeichen (') oder (") eingeschlossen werden.

Moduldefinitionsdateien

Beispiel einer Moduldefinitionsdatei

Nachfolgend eine typische Moduldefinitionsdatei:

```
NAME            MYPROG
DESCRIPTION        'Myprog Version 1.00'
EXETYPE            WINDOWS 3.1
CODE            MOVEABLE DISCARDABLE
DATA            PRELOAD MOVEABLE MULTIPLE
RC              MYPROG
HEAPSIZE        1024
STACKSIZE       8192
```

Um eine .DLL anstatt einer .EXE zu erzeugen, ersetzten Sie einfach den Befehl NAME MYPROG durch den Befehl LIBRARY MYDLL.

NAME gibt den Namen des Programm-Moduls an, normalerweise den der ausführbaren Datei, und zeigt an, daß eine .EXE erzeugt werden soll.

LIBRARY gibt den Namen des Programm-Moduls an, normalerweise den der ausführbaren Datei, und zeigt an, daß eine .DLL erzeugt werden soll.

DESCRIPTION ermöglicht die Angabe eines beschreibenden Zeichenstrings, der in das Programm

eingefügt wird.

EXETYPE gibt an, unter welchem Betriebssystem das Programm laufen soll.

CODE setzt die Standardattribute der Codesegmente. MOVEABLE legt fest, ob ein Segment im Speicher verschoben werden kann. DISCARDABLE legt fest, ob ein Codesegment aus dem Speicher entfernt werden kann, wenn weitere Speicheranforderungen erfüllt werden sollen.

DATA setzt die Standardattribute der Datensegmente. PRELOAD zeigt an, daß das Segment beim Programmstart geladen wird. MOVEABLE legt fest, ob ein Segment im Speicher verschoben werden kann. MULTIPLE bestimmt, daß DGROUP für jede Instanz des Programms kopiert werden muß.

RC gibt den Namen der Ressource-Datei für das Programm an.

HEAPSIZE setzt die Größe des lokalen Heap in Bytes für das Programm.

STACKSIZE setzt die Größe des Stack in Bytes für das Programm.

Der BLILIB Import-Bibliotheksmanager

Importbibliotheken

Die Verwendung von BLILIB

Der BLILIB Import-Bibliotheksmanager

Importbibliotheken

Importbibliotheken sind einfach Bibliotheksdateien, die IMPORTS Definitionen für jede der exportierten Symbole einer .DLL enthalten. Sie können automatisch aus einer .DLL erzeugt werden mit Hilfe eines Import-Bibliotheksmanagers.

Importbibliotheken werden von Programmen benötigt, die Referenzen in DLLs auflösen wollen. Ohne Importbibliothek muß eine Referenz auf eine dynamische Linkroutine in einer DLL entweder mit der IMPORTS-Anweisung in der .DEF-Datei zum Programm oder explizit im Programm kodiert angegeben werden.

Der BLILIB Import-Bibliotheksmanager

Die Verwendung von BLILIB

Zur Erzeugung von Importbibliotheken oder Importdefinitionen zur Verwendung mit Windows- oder DOS-Extenderprogrammen dient der Import-Bibliotheksmanager BLILIB.

Beim Lesen einer DLL-Datei verwendet BLILIB nur die exportierten Namen des neuen .EXE-Headers und ignoriert den Rest der DLL.

Beim Lesen einer Moduldefinitionsdatei wertet BLILIB nur die folgenden drei Anweisungen aus:

LIBRARY
EXPORTS
INCLUDE

Jeder weitere Text wird ignoriert.

Die Syntax eines Aufrufs von BLILIB auf Kommandozeilenebene lautet:

```
BLILIB /I{D | E} inputdatei /O{I | L} outputdatei [optionen]
```

wobei /Ix die Eingabedatei anzeigt:

/ID inputdatei	Eine .DLL-Datei mit exportierten Namen
/IE inputdatei	Eine Moduldefinitionsdatei mit einer Export-Definition

/Ox zeigt die Ausgabedatei an:

/OI outputdatei	Eine Moduldefinitionsdatei mit einer Import-Definition
/OL outputdatei	Eine .LIB-Datei mit Import-Definitionen

und als Optionen:

/MH oder /?	Zeigt Hilfe und Syntax der Befehlszeile an
/MV	Verbose - zeigt Meldungen während des Ablaufs an
/NL	Verhindert die Anzeige des Copyright-Logos
/NP	Verhindert den Prompt, wenn Ausgabedatei bereits existiert
/SM	Behandelt alle Symbolnamen als Mixed-Case, d.h. Groß-und Kleinschreibung wird unterschieden.

Optionen können in Groß- oder Kleinschrift angegeben werden. Werden keine Dateierweiterungen angegeben, fügt BLILIB die Standard-Dateierweiterungen an: .DLL für DLLs, .LIB für Importbibliotheken und .DEF für Moduldefinitionsdateien.

Ist die Eingabedatei eine DLL, werden alle Exporte aus der DLL in die Importbibliothek abgelegt. Wenn nur eine Teilmenge an exportierten Elementen aus der DLL in die Importbibliothek abgelegt werden soll, können Sie eine Moduldefinitionsdatei erzeugen, die nur diese Exporte beinhaltet, und verwenden Sie diese als Eingabedatei.

BLILIB sucht in dem Pfad, auf den die Umgebungsvariable LIB verweist, wenn irgendeine Datei nicht im aktuellen Verzeichnis gefunden wird.

Beispiel:

```
BLILIB /ID mylib /OL mylib /NL
```

Dieser Befehl erzeugt die Importbibliothek MYLIB.LIB aus der dynamischen Linkbibliothek MYLIB.DLL und zeigt kein Copyright-Logo an.

Linken von 16-Bit-Windows-Programmen

Im ersten Schritt werden die Blinker Link-Skriptdatei und optionale Moduldefinitionsdateien, einschließlich des RC Moduldefinitionsbefehls, vorbereitet, um den letzten Resource-Compilerlauf zu ersetzen. Sehen Sie hierzu auch in Kapitel 2 ‘Schnell-Einstieg’ in den entsprechenden Abschnitten für Ihren Compiler die Beispiele von Link-Skript- und Moduldefinitionsdateien.

Sie können sich wahlweise auch eine Batchdatei oder ein Icon für den Blinker mit seiner Kommandozeile, die Namen und Pfad der Link-Skriptdatei enthält, anlegen.

Compilieren Sie Ihren Quelltext für Windows, wie in Ihrem Compiler-Handbuch beschrieben.

Starten Sie den Resource-Compiler, wie in Ihrem Compilerhandbuch erklärt, zur Erzeugung aller benötigten Ressourcen.

Starten Sie Blinker mit der oben erzeugten Link-Skriptdatei mittels Batchdatei oder Icon oder mit Aufruf über die DOS-Befehlszeile, um die .EXE-Datei zu erzeugen und hängen Sie alle Ressourcen an das Ende der .EXE-Datei an.

Eine 16-Bit Windows-Moduldefinitionsdatei sieht typischerweise folgendermaßen aus:

```
; TEST.DEF
NAME            TEST
DESCRIPTION     'kleines 16-Bit Windows-Programm'
EXETYPE         WINDOWS 3.1
CODE            MOVEABLE DISCARDABLE
DATA            PRELOAD MOVEABLE MULTIPLE
RC              TEST
HEAPSIZE        1024
STACKSIZE       8192
EXPORTS         ....
```

Linken von 32-Bit-Windows-Programmen

Verwenden Sie hier den gleichen Ablauf wie beim Linken eines 16-Bit Windows-Programmes, mit folgenden kleinen Änderungen:

- Der Compiler und seine Laufzeitbibliothek müssen die Erzeugung von 32-Bit Windows-Programmen unterstützen.
- Einige Moduldefinitionsbefehle unterscheiden sich leicht oder werden nicht beim Linken von 32-Bit Windows-Programmen verwendet.
- Einige Linkerparameter müssen in Moduldefinitionsbefehle umgesetzt werden, z.B.: /SUBSYSTEM und /VERSION.

Eine 32-Bit Windows-Moduldefinitionsdatei sieht typischerweise folgendermaßen aus:

```
; TEST.DEF
NAME            TEST
DESCRIPTION     'kleines 32 Bit Windows-Programm'
VERSION         1.0
EXETYPE         NT SUBSYSTEM WINDOWS
RC              TEST
HEAPSIZE        65536, 4096
```

```
STACKSIZE        65536, 4096
EXPORTS          ....
```

Moduldefinitionen - eine Quick-Referenz

APPLOADER	Standard-Ladeprogramms von Windows durch ein benutzerdefiniertes Ladeprogramm ersetzen
CODE	Setzen der Voreinstellung für die CODE-Segmentattribute
DATA	Setzen der Voreinstellung für die DATA-Segmentattribute
DESCRIPTION	Einfügen eines beschreibenden Textes in das Programm
EXETYPE	Identifizieren des Ziel-Betriebssystems
EXPORTS	Definieren exportierter Funktionen
HEAPSIZE	Größe des lokalen Heap in Bytes setzen
IMPORTS	Definieren importierter Funktionen
INCLUDE	Einfügen einer anderen Datei in die .DEF-Datei
LIBRARY	Erzeugen und Benennen einer DLL
NAME	Erzeugen und Benennen eines Anwendungsprogramms
PROTMODE	Angabe eines Windows-basierten Protected Mode-Programms
RC	Anfügen der Resourcedatei an die EXE-Datei
REALMODE	Angabe eines Windows-basierten Real Mode-Programms
SECTIONS	Attribute bestimmter Abschnitte setzen
SEGMENTS	Attribute bestimmter Segmente setzen
STACKSIZE	Angabe der Stackgröße in Bytes
STUB	Hinzufügen eines DOS-Stub an den Beginn einer Programmdatei
VERSION	Versionsnummer eines 32-Bit Programmes setzen

Moduldefinitionsbefehle

Moduldefinitionsbefehle stehen für 16-Bit und 32-Bit Windows-Programme gleichermaßen zur Verfügung, solange nicht anders angegeben.

APPLOADER (16 Bit)

CODE

DATA

DESCRIPTION

EXETYPE

EXPORTS

HEAPSIZE

IMPORTS

INCLUDE

LIBRARY (Moduldefinitionsbefehl)

NAME

PROTMODE (16 Bit)

RC (Moduldefinitionsbefehl)

REALMODE (16 Bit)

SECTIONS

SEGMENTS

STACKSIZE

STUB

VERSION (32 Bit)

Moduldefinitionsbefehle

APPLOADER (16 Bit)

Zweck	Ersetzt das voreingestellte Windows-Ladeprogramm durch ein benutzerdefiniertes
Syntax	APPLOADER 'ladername'
Beispiel	APPLOADER 'MYLOADER'

Dieser optionale Befehl teilt Blinker mit, daß ein extern definiertes Ladeprogramm verwendet werden soll. Dabei wird dem übergebenen Namen das Präfix LOADER_ vorangestellt.

Dieser Befehl wurde zur Unterstützung selbstladender FORTRAN-Programme beigefügt und muß vor allen SEGMENTS-Anweisungen stehen.

Moduldefinitionsbefehle

CODE

Zweck	Legt die Voreinstellung für die CODE-Segmentattribute fest
Syntax	CODE [attribut [attribut ...]]
Beispiel	CODE PRELOAD EXECUTEREAD

Die über den Befehl CODE gesetzten Standardattribute können mit Hilfe des Befehls SEGMENTS überschrieben werden. Die Attribute können in beliebiger Reihenfolge angegeben werden. Dieser Befehl sollte in der Definitionsdatei vor sämtlichen Instanzen des SEGMENTS-Befehls, die auf Codesegmente verweisen, stehen.

Eine vollständige Auflistung der anwendbaren Attribute finden Sie unter dem Eintrag SEGMENTS weiter unten in diesem Kapitel.

Moduldefinitionsbefehle

DATA

Zweck	Setzt die Voreinstellung für die DATA-Segmentattribute.
Syntax	DATA [attribute [attribute ...]]
Beispiel	DATA PRELOAD

Die über den Befehl DATA gesetzten Standardattribute können mit Hilfe des Befehls SEGMENTS überschrieben werden. Die Attribute können in beliebiger Reihenfolge angegeben werden. Dieser Befehl sollte in der Definitionsdatei vor sämtlichen Instanzen des SEGMENTS-Befehls, die auf Datensegmente verweisen, aufgeführt werden.

Eine vollständige Auflistung der anwendbaren Attribute finden Sie unter dem Eintrag SEGMENTS weiter unten in diesem Kapitel.

Moduldefinitionsbefehle

DESCRIPTION

Zweck	Fügt einen beschreibenden Text in ein Programm ein
Syntax	DESCRIPTION 'Beschreibungstext'
Beispiel	DESCRIPTION 'Professionelle Version 1.00'

Über diesen Befehl wird ein bis zu 255 Zeichen langer Textstring in eine .EXE- oder .DLL-Datei eingefügt. Beachten Sie dabei, daß diese Information standardmäßig nicht vom erstellten Programm verwendet wird.

Dieser Befehl ist sehr gut geeignet, um Copyright- oder Versionsinformationen in ein Programm einzufügen.

Moduldefinitionsbefehle

EXETYPE

Zweck	Kennzeichnen des Ziel-Betriebssystems
Syntax 16	EXETYPE WINDOWS [major[.minor]] | OS2 | DOS
Beispiel 16	EXETYPE WINDOWS 3.1
Syntax 32	EXETYPE NT [SUBSYSTEM subsystem [n[.n]]]
Beispiel 32	EXETYPE NT SUBSYSTEM WINDOWS 4.0

Diese Befehl gibt an, unter welchem Betriebssystem das Programm später laufen soll. Die Voreinstellung ist der DOS-Extender, deshalb muß dieser Befehl für alle Windows-Programme angegeben werden.

Bei der Erzeugung von Windows-Programmen ist es wichtig, die Version von Windows anzugeben, damit sichergestellt ist, daß das Programm auch wirklich ausgeführt werden kann. Die voreingestellte Version ist 3.1, wenn nicht anders angegeben.

Für ein 32-Bit Windows-Programm kennzeichnet subsystem eine der folgenden Eigenschaften:

CONSOLE zeigt an, daß das Programm im Win32 Zeichenmodus betrieben wird.

NATIVE zeigt an, daß es sich bei dem Programm um einen Device-Treiber für Windows NT handelt.

PHARLAP zeigt an, daß das Programm den PharLap 32 Bit Extender (TNT) verwenden soll.

POSIX zeigt an, daß das Programmes unter dem POSIX Subsystem von Windows NT betrieben werden soll.

WINDOWS, zeigt an, daß das Programm keine Console benötigt, weil es seine eigenen Fenster erzeugt. Die Win32s-Betriebssysteme können nur WINDOWS-Anwendungen ausführen.

Moduldefinitionsbefehle

EXPORTS

Zweck	Definiert exportierte Funktionen
Syntax	EXPORTS entryname [=internalname] [@ord [nametable]] [DATA] [NODATA]
Beispiele	EXPORTS Initialise @1 NONAME ResetRdr RESIDENTNAME EndRdr = ReaderEnd

Dieser Befehl macht die aufgeführten Funktionen in einer DLL (die voreingestellt ‘hidden’, d.h. unsichtbar sind), anderen ausführbaren Dateien zur Laufzeit zugänglich.

entryname definiert den Namen der Funktion, so wie er in anderen Programmen bekannt ist, d.h. wie er exportiert wird.

internalname definiert den aktuellen Namen der Funktion, so wie er innerhalb des exportierenden Programm erscheint. Standardmäßig sind entryname und internalname gleich.

ord definiert die Ordnungsnummer (den Index) einer Funktion innerhalb der Moduldefinitionstabelle, d.h. die Nummer, auf die eine Importdefinition zur Identifikation innerhalb der DLL verweist.

nametable ist eines der beiden optionalen Schlüsselwörter RESIDENTNAME oder NONAME, mit denen festgelegt wird, ob der Funktionsname in der residenten oder nicht residenten Namenstabelle eingetragen wird, oder ob er zur Linkzeit verworfen werden soll. Standardmäßig kann ein Symbol nur über die Ordnungsnummer importiert werden, auch wenn der Name in der nicht-residenten Namenstabelle gespeichert ist.

NONAME bedeutet, daß das Symbol kann nur über die angegebene Ordnungsnummer importiert werden kann, da der Name nicht gespeichert ist.

RESIDENTNAME speichert den Namen in der residenten Namenstabelle und erlaubt den Import einer Funktion direkt über den Namen.

DATA ist nur bei 32-Bit-Programmen gültig und zeigt an, daß es sich bei dem exportierten Element um ein Datenelement handelt. Das Datenelement im Client-Programm muß mit der extern __declspec(dllimport)-Anweisung deklariert worden sein.

NODATA zeigt an, daß keine statischen Daten in der Funktion verwendet werden, daß die Funktion also das aktuelle Datensegment verwendet.

Moduldefinitionsbefehle

HEAPSIZE

Zweck	Legt die Größe des lokalen Heap in Bytes fest
Syntax 16	HEAPSIZE größe | MAXVAL
Beispiel 16	HEAPSIZE 3000
Syntax 32	HEAPSIZE reserviertgröße [, commitgröße]
Beispiel 16	HEAPSIZE 65536,4096

Über diesen Befehl wird die Größe des lokalen Heap eines Programmes in Bytes gesetzt. Bei 32-Bit Programmen kann optional auch eine Commitgröße mit angegeben werden.

Bei 16-Bit Windows-Programmen wird der Heap im Standard-Datensegment nach DGROUP erzeugt. Seine maximale Größe wird also durch die Größe von DGROUP beeinflußt. Mit dem Schlüsselwort MAXVAL können Sie die Größe des Heap bei 16-Bit Programmen auf 64 KB minus der Größe von DGROUP, d.h. auf die maximal mögliche Größe setzen.

Moduldefinitionsbefehle

IMPORTS

Zweck	Definiert importierte Funktionen
Syntax	IMPORTS [internalname =] modname.eintrag
Beispiele	IMPORTS READER.5 Initialise = READER.1 READER.EndRdr Reset = READER.ResetRdr

Dieser Befehl definiert die Namen und die Position von Funktionen, die zur Verwendung im aktuellen Programm importiert werden sollen. Dabei ist für jede zu importierende Funktion eine eigene Definition erforderlich.

internalname ist die Funktion, die im importierenden Programm eingesetzt werden soll.

modname ist der Dateiname des exportierenden Programms, in dem die Funktion enthalten ist.

eintrag kann entweder den Namen oder die Ordnungsnummer der Funktion enthalten, so wie er im Programm modname definiert wurde. Handelt es sich hierbei um eine Ordnungsnummer, die in der EXPORT-Anweisung gesetzt wurde, dann muß internalname angegeben werden.

Moduldefinitionsbefehle

INCLUDE

Zweck	Fügt eine andere Datei in die .DEF-Datei ein
Syntax	INCLUDE dateiname
Beispiel	INCLUDE OTHERFILE

Dieser Befehl fügt den Inhalt einer angegebenen Datei genau an der Stelle in die .DEF-Datei ein, an der der Befehl INCLUDE angegeben ist. Die einzufügende Datei muß gültige Moduldefinitionsanweisungen enthalten. Eingefügte Dateien können bis zu vier Ebenen tief verschachtelt werden.

Moduldefinitionsbefehle

LIBRARY (Moduldefinitionsbefehl)

Zweck	Erzeugt und benennt eine DLL
Syntax 16	LIBRARY [libname] [PRIVATELIB] [WINDOWAPI]
Syntax 32	LIBRARY [libname] [BASE = adresse]
Beispiel	LIBRARY mydll

Dieser Befehl setzt den internen Namen und zeigt an, daß die zu erzeugende ausführbare Datei eine Dynamische Linkbibliothek (DLL) sein wird. Wenn dieser Befehl eingesetzt wird, muß die Anweisung LIBRARY an erster Stelle stehen. Wurde kein Name angegeben, dann wird der Name der ausführbaren Datei ohne dessen Erweiterung verwendet.

Das Schlüsselwort PRIVATELIB zeigt Windows an, daß zu jeder Zeit nur ein Programm diese DLL verwenden darf.

Das Schlüsselwort WINDOWAPI ist Standard und zeigt an, daß die DLL eine Windows- oder Protected Mode-Anwendung ist, die nur unter Windows oder in einer Protected Mode-Umgebung ausgeführt werden kann. Für Windows bedeutet dies, daß auf das RC-Kommando oder den Resource-Compiler verzichtet werden kann, wenn das Programm keine Ressourcen besitzt.

Bei 32-Bit Windows-Programmen wird das Schlüsselwort BASE dazu verwendet, um die Basisadresse festzulegen, an der das Betriebssystem die DLL laden soll. Voreinstellung ist die Adresse 0x10000000.

Moduldefinitionsbefehle

NAME

Zweck	Erzeugt und benennt ein Anwendungsprogramm
Syntax 16	NAME [appname] [apptype] [NEWFILES]
Syntax 32	NAME [appname] [BASE = adresse]
Beispiel	NAME myapp

Diese Anweisung kennzeichnet die ausführbare Datei als ein Programm (im Gegensatz zu einer DLL). Dieser Befehl muß - wenn verwendet - an erster Stelle stehen und kann nicht zusammen mit dem Befehl LIBRARY verwendet werden. Wird sowohl auf die Angabe von NAME als auch auf LIBRARY verzichtet, dann gilt NAME als Voreinstellung.

appname ist der Name der Applikation, unter dem sie dem Betriebssystem bekannt ist. Dieser Name darf nicht mehr als 255 Zeichen beinhalten. Wurde kein Name angegeben, dann wird standardmäßig der Basisname der ausführbaren Datei verwendet.

apptype enthält eines der drei optionalen Schlüsselwörter:

WINDOWAPI. Dieses Schlüsselwort wird standardmäßig verwendet. Es erzeugt eine Windows-basiertes Programm. Dieses Programm nutzt das Windows-API und muß unter Windows ausgeführt werden.

WINDOWCOMPAT. Über diese Option wird ein Programm im Zeichenmodus erzeugt. Solch ein Programm kann unter Windows in einem Textfenster ausgeführt werden.

NOTWINDOWCOMPAT. Über diese Option wird ein Programm im Zeichenmodus erzeugt. Dieses Programm benötigt den gesamten Bildschirm. Es ist also nicht möglich, solch ein Programm in einem Textfenster unter Windows ausführen.

Über das Schlüsselwort NEWFILES wird dem Ladeprogramm mitgeteilt, daß das Programm ein installierbares Dateisystem verwenden könnte. Das Synonym LONGNAMES wird aus Kompatibilitätsgründen unterstützt.

Bei 32-Bit Windows-Programmen wird das Schlüsselwort BASE dazu verwendet, um die Basisadresse festzulegen, an der das Betriebssystem die DLL laden soll. Voreinstellung ist die Adresse 0x400000. adresse wird als vorzeichenloser Integer angegeben.

Moduldefinitionsbefehle

PROTMODE (16 Bit)

Zweck Gibt ein Windows-basiertes Protected Mode-Programm an

Syntax PROTMODE

Dieser Befehl gibt an, daß das entsprechende Programm nur unter Windows im Protected Mode ausgeführt werden darf.

Moduldefinitionsbefehle

RC (Moduldefinitionsbefehl)

Zweck Legt fest, daß eine Resource-Datei in der .EXE-Datei eingefügt wird.

Syntax RC [-m] resdateiname[.RES]

Dieser Befehl gibt an, daß die Resource-Datei resdateiname in der .EXE-Datei eingefügt wird.

Durch die Option -m wird Blinker dazu veranlaßt, das Mehrinstanzenflag zu setzen. Andere Optionen, die üblicherweise mit dem Resource-Compiler gesetzt werden, können durch die Verwendung von Moduldefinitionsbefehlen gesetzt werden.

Der Resource-Compiler WRC von WATCOM benötigt die Option -zm, um MS-Format 16-Bit-Ressourcen auszugeben, die mit dem Blinker-Befehl RC verwendet werden können. WRC muß für den zweiten Resource-Compilerlauf mit 32-Bit-Programmen verwendet werden, da er ein nicht standardisiertes 32-Bit-Resource-Dateiformat verwendet.

Moduldefinitionsbefehle

REALMODE (16 Bit)

Zweck Gibt ein Windows-basiertes Real Mode-Programm an

Syntax REALMODE

Dieser Befehl gibt an, daß das entsprechende Programm unter Windows im Real und im Protected Mode ausgeführt werden kann.

Moduldefinitionsbefehle

SECTIONS

Dieser Befehl ist funktional gleich zum SEGMENTS-Befehl und wird von 32-Bit Windows-Programmen verwendet, um Attribute für spezielle Bereiche zu setzen.

Moduldefinitionsbefehle

SEGMENTS

Zweck	Setzt Attribute für bestimmte Segmente
Syntax	SEGMENTS [']segname['] [CLASS 'klassenname'] [attribut [attribut ...]]
Beispiel	SEGMENTS seg1 CLASS 'mycode' EXECUTEONLY seg2 CLASS 'mydata' PRELOAD

Mit dem Befehl SEGMENTS können die Attribute für ein oder mehrere Segmente segmentweise gesetzt werden. Dadurch werden die über die Befehle CODE und DATA angegebenen Voreinstellungen überschrieben. Die Reihenfolge der angegebenen Segmente bestimmt ihre Reihenfolge in der Ausgabedatei. Falls CLASS "klassenname" nicht angegeben wird, ist ‘CODE’ die Vorgabe.

Bei 32-Bit Windows-Programmen können diese Attribute eins oder mehrere aus EXECUTE, READ, SHARED und WRITE sein.

Bei 16-Bit Windows-Programmen können folgende Attribute angegeben werden (voreingestellte Werte sind fettgedruckt):

discard [NONDISCARDABLE | DISCARDABLE]

Diese Option wird nur für die Befehle CODE und SEGMENTS benutzt. Sie legt fest, ob ein Codesegment in einem Real Mode Windowsprogramm aus dem Speicher entfernt werden darf, wenn weitere Speicheranforderungen erfüllt werden müssen.

execute [EXECUTEREAD | EXECUTEONLY]

Diese Option wird nur für die Befehle CODE und SEGMENTS benutzt. Sie legt fest, ob ein Codesegment ebenso gelesen wie ausgeführt werden kann. Die Option EXECUTEREAD muß für ein Programm angegeben werden, das unter einem Debugger ausgeführt werden soll, der CodeView-Informationen verarbeitet.

instance [MULTIPLE | SINGLE | NONE]

Diese Option wird nur für den Befehl DATA verwendet. Sie wirkt sich auf die Shared-Attribute von DGROUP, dem Standard-Datensegment aus. Diese Option und die Shared-Attribute beeinflussen sich gegenseitig.

MULTIPLE (Voreinstellung für Programme) gibt an, daß DGROUP für jede Instanz des Programms

kopiert werden soll.

SINGLE (Voreinstellung für DLLs) gibt an, daß sich alle Instanzen einer DLL oder eines Programms eine DGROUP teilen müssen.

NONE teilt dem Ladeprogramm mit, daß kein Speicher für eine DGROUP belegt werden soll. Diese Option wird verwendet, wenn eine DLL keine Daten enthält und daher die DGROUP des Programms nutzt.

Diese Option und die Shared-Attribute der Datensegmente beeinflussen sich gegenseitig.

load [LOADONCALL | PRELOAD]

Diese Option wird für die Befehle CODE, DATA und SEGMENTS verwendet. Sie legt fest, wann ein Segment geladen wird. LOADONCALL bewirkt, daß das Segment erst geladen werden soll, wenn auf dieses Segment zugegriffen wird. Es soll nur dann geladen werden, wenn es nicht bereits geladen wurde. PRELOAD gibt an, daß das Segment beim Programmstart geladen werden soll.

moveable [MOVEABLE | FIXED]

Diese Option wird für die Befehle CODE, DATA und SEGMENTS eingesetzt. Sie legt fest, ob ein Segment eines Real Mode-Windows-Programms im Speicher verschoben werden kann. In einem DOS-Extender-Programm erzwingt das Attribut FIXED das Laden des Segmentes in den konventionellen Speicher, auf den sowohl im Real Mode als auch im Protected Mode zugegriffen werden kann. Konventioneller Speicher ist eine begrenzte Ressource, daher sollte diese Option wirklich nur dort eingesetzt werden, wo sie unbedingt nötig ist, z.B. für bimodale Interrupt-Behandlungsroutinen.

read [READWRITE | READONLY]

Diese Option wird nur für die Befehle DATA und SEGMENTS eingesetzt. Sie legt die Zugriffsrechte auf ein Datensegment fest.

shared [NONSHARED | SHARED]

Diese Option wird nur für Windows-Arbeitssitzungen im Real Mode verwendet. Sie legt fest, ob sich alle Instanzen eines Programmes EXECUTEREAD- und READWRITE-Segmente teilen können.

NONSHARED (Voreinstellung für Programme) gibt an, daß das Segment für jeden Prozeß gesondert geladen werden muß. Ein alternatives Schlüsselwort ist IMPURE.

SHARED (Voreinstellung für DLLs) gibt an, daß nur eine Kopie des Segments geladen werden soll, die sich dann alle Prozesse teilen müssen, die auf ein Programm oder eine DLL zugreifen. Über diese Einstellung wird Speicherplatz eingespart. Sie kann für Code verwendet werden, der sich nicht selbst ändert. Das alternative Schlüsselwort ist PURE. Diese Option und die instance-Attribute der Datensegmente beeinflussen sich gegenseitig.

Moduldefinitionsbefehle

STACKSIZE

Zweck	Setzt die Größe des Stack in Bytes
Syntax 16	STACKSIZE größe
Beispiel 16	STACKSIZE 4096
Syntax 32	STACKSIZE reserviertgröße [, commitgröße]
Beispiel 32	STACKSIZE 65536,4096

Dieser Befehl setzt die Größe des Stack für ein Programm in Bytes und gibt bei 32-Bit-Programmen optional auch eine Commitgröße in Bytes mit an.

Moduldefinitionsbefehle

STUB

Zweck	Fügt ein DOS-Stub am Beginn der Programmdatei ein
Syntax	STUB 'dateiname' | NONE
Beispiel	STUB 'MYSTUB.EXE'

Der in das Programm eingefügte Stub wird immer dann aufgerufen, wenn das Programm unter DOS gestartet wird. Normalerweise zeigt es eine Warnung an und beendet die Ausführung des Programms.

In jedes mit Blinker erzeugte Windows-.EXE-Programm wird automatisch ein Stub-Programm eingefügt, das diese Aufgabe erfüllt, wenn kein STUB-Befehl angegeben wurde.

Wenn sich die angegebene Datei nicht im aktuellen Verzeichnis befindet, dann werden die über die Umgebungsvariable PATH angegebenen Verzeichnisse durchsucht.

Über das Schlüsselwort NONE kann man angeben, daß die .EXE-Datei ohne Stub-Programm erzeugt werden soll. Dadurch wird die ausführbare Datei etwas kleiner. Wenn die .EXE-Datei aber vom DOS-Prompt her ausgeführt wird, hängt sich das System meistens auf. Die Verwendung dieser Einstellung wird daher nicht empfohlen.

Moduldefinitionsbefehle

VERSION (32 Bit)

Zweck	Setzt die Versionsnummer eines 32-Bit Windows-Programmes
Syntax	VERSION haupt[.unter]
Beispiel	VERSION 1.1

Dieses Kommando setzt die Versionsnummer im Header eines 32-Bit Windows-Programmes. Vorgabewert ist 0.0.

Spezielle Features

.EXE- und .DLL-Dateikomprimierung

Die Swap-Funktion des Blinkers

Allgemeine Debugger-Unterstützung

Debuggen von Real Mode Overlay-Programmen

Unterstützung für Profiler

Erzeugen von Demo-Versionen

Seriennummern-Verschlüsselung

Der Blinker Error-Handler

.EXE- und .DLL-Dateikomprimierung

Übersicht

Einschalten der Komprimierung

Komprimieren von Windows-Programmen

Komprimieren von DOS-Extender-Programmen

Komprimieren von DOS-Real Mode-Programmen

.EXE- und .DLL-Dateikomprimierung

Übersicht

Das Komprimieren von .EXE- und .DLL-Dateien mit Blinker für den endgültigen Linkvorgang vor der Auslieferung ermöglicht, in gleicher Weise wie durch herkömmliche Komprimierungs- und Archivierungsprogramme, Vertriebskosten zu senken, bietet jedoch den entscheidenden Vorteil, daß zur Installations- oder Laufzeit kein externes Dekomprimierungsprogramm benötigt wird. Die Tatsache, daß es sich um ein komprimiertes Programm handelt, bleibt dadurch vollständig transparent.

Zusätzlich wird der Datenfluß im Netzwerk merkbar reduziert, wenn .EXE- und .DLL-Dateien auf einem Netzwerk-Server abgelegt sind, da ja anstelle der kompletten dekomprimierten Datei nur die komprimierte Datei im Netz übertragen wird.

Als weitereren Vorzug bietet die Komprimierung eine gute Möglichkeit der Verschlüsselung: Da mit dem Blinker komprimierte Programme für Disassembler und Decompiler verschlüsselt sind, wird ein guter Schutz Ihres Entwicklungsaufwands und aller Innovationen in Ihrem Programm geboten.

Blinker kann .EXE-Dateien für 16-Bit-Windows-, DOS und den DOS-Extender, wie auch DLLs für Windows und für den DOS-Extender transparent komprimieren. Bei DOS-Programmen für den Real Mode oder DOS-Extender-Programmen für den Dual Mode werden Overlays nicht komprimiert, da sich Aufwand und Nutzen in etwa gegeneinander aufheben, wie weiter unten im Abschnitt ‘Komprimieren von DOS-Real Mode-Programmen’ erläutert.

Der Dekomprimierungsvorgang benötigt bis zu 100 KBytes Hauptspeicher zusätzlich zum normalen Speicherbedarf des Programmes. Für ein DOS-Real Mode-Programm wird dieser Speicher nach dem Programmstart nicht mehr benötigt, geht also nicht verloren. Bei Programmen unter Windows und unter dem DOS-Extender jedoch werden Segmente nur auf Anforderung geladen, d.h.: dieser Speicherbereich steht für den Rest des Programmes nicht zur Verfügung.

Bei Programmen für Windows oder den DOS-Extender wird jedes Segment separat komprimiert, wodurch es möglich ist, daß es erst zur Laufzeit bei Bedarf geladen bzw. nicht geladen (verworfen) werden kann. Die erzielte Kompressionsrate ist daher nicht so hoch wie bei manchen Archivierungsprogrammen, die eine gesamte .EXE-Datei in einem Block komprimieren. Dafür geht bei der Verwendung von externen Archivierungsprogrammen der Vorteil einer transparenten Dekomprimierung zur Laufzeit verloren.

Für bestmögliche Komprimierung von Windows- und DOS-Extender-Programmen sollten die einzelnen Segmente eines Programmes so groß wie möglich gehalten und die Optionen PACKCODE und PACKDATA - falls nicht bereits gesetzt - eingeschaltet werden. Mehr zu diesen Befehlen in Kapitel 9. Die Größe der einzelnen Segmente kann der .MAP-Datei, die über den Linkskriptbefehl MAP zur Linkzeit erzeugt wird, entnommen werden.

Komprimierung von Visual Basic DOS-Programmen, die CHAIN verwenden, wird momentan nicht unterstützt, da die Laufzeitbibliothek versucht, Daten aus den mit CHAIN verketteten Dateien zu lesen..

.EXE- und .DLL-Dateikomprimierung

Einschalten der Komprimierung

Um die Komprimierung einzuschalten, fügen Sie einfach den Befehl

BLINKER EXECUTABLE COMPRESS [<Faktor>]

in das Linkskript ein. Der Vorgabewert des optionalen Kompressionsfaktors liegt bei 5, er kann jedoch auf Werte von 1 bis 9 eingestellt werden, um mehr Wert auf Kompressionsrate oder aber auf Komprimierungs- bzw. Dekomprimierungsgeschwindigkeit zu legen.

Der Wert 1 erzielt die maximale Kompression, was zwar die längste Linkzeit bedeutet, jedoch auch die kleinstmögliche Größe der .EXE-Datei zur Folge hat. Der Wert 9 bringt schnellstmögliche Linkzeit mit immer noch vernünftiger Kompression.

Aufgrund des Algorithmus ändert sich die Zeitdauer für das Dekomprimieren bei verschiedenen Kompressionsfaktoren nicht merklich. Testen Sie jedoch verschiedene Optionen mit Ihrem Programm, um das bestmögliche, gewünschte Verhalten zu erzielen.

Der zusätzliche Verarbeitungsaufwand für das Komprimieren eines Programmes macht sich in einer langsameren Linkgeschwindigkeit bemerkbar, daher sollte der Komprimierungsbefehl typischerweise erst vor dem endgültigen Linklauf eines Programmes oder der Fertigstellung einer Testversion angegeben werden. Außer der erhöhten Linkzeit gibt es jedoch keinen Grund, die Komprimierung nicht zu jedem gewünschten Zeitpunkt zu verwenden.

Hinweis zu CA-Clipper: Das Einschalten der Komprimierung schaltet automatisch das inkrementelle Linken ab, da diese beiden Merkmale nicht gleichzeitig verwendet werden können. Dies wäre auch ein Grund, die Komprimierung nicht dauernd einzusetzen.

.EXE- und .DLL-Dateikomprimierung

Komprimieren von Windows-Programmen

Zur Linkzeit wird eine kleine Dekomprimierungsroutine als erstes Segment der .EXE- oder .DLL-Datei angefügt. Jedes Segment dieser Datei wird dann separat komprimiert, damit es zur Laufzeit nach Bedarf geladen bzw. nicht geladen werden kann.

Standardmäßig werden die meisten Windows-Programme so erzeugt, daß alle Codesegmente auf PRELOAD in der Moduldefinitionsdatei gesetzt sind. Dies bewirkt zur Ladezeit die größte Verzögerung, da erst alle Codesegmente geladen und dekomprimiert werden, bevor das Programm startet. Wir empfehlen, dieses Attribut nicht zu setzen, solange es keinen speziellen Grund gibt, alle Segmente auf PRELOAD zu setzen. Ist das Attribut PRELOAD nicht gesetzt, können Codesegmente erst bei Bedarf geladen werden, wodurch sich die Ladezeit großer komprimierter Programme beim Programmstart merkbar verringert.

Hinweis: Wenn es beispielsweise nötig ist, einen externen Resource-Compiler, wie etwa RC.EXE, für eine .EXE-Datei, die mit Blinker erzeugt wurde, einzusetzen, stellen Sie sicher, daß die Option -k (Segmentreihenfolge beibehalten) des Resource-Compilers verwendet wird, andernfalls wird das

Programm nicht geladen.

.EXE- und .DLL-Dateikomprimierung

Komprimieren von DOS-Extender-Programmen

Auch hier wird die .EXE- oder .DLL-Datei zur Linkzeit Segment für Segment komprimiert, der Lader des DOS-Extenders erkennt jedoch zur Ladezeit automatisch ein komprimiertes Segment.

Da der DOS-Extender Speicher in relativ großen Blöcken alloziert (siehe BLINKER EXTMEM BLOCK in Kapitel 9), scheint es, als ob der für ein DOS-Extender-Programm verfügbare Speicher, wie er von einer Bibliotheksfunktion oder einer der Blinker-Speicherfunktionen zurückgegeben wird, keinen oder einen größeren Unterschied bis hin zur Größe EXTMEM BLOCK zeigt. Weitere Details finden Sie in Beschreibung des Befehls BLINKER EXTMEM BLOCK nnn in der Online-Hilfe.

Zum genaueren Testen des Speicher-Overheads können Sie temporär die Blockgröße über den Befehl BLINKER EXTMEM BLOCK reduzieren, sollten diese Einstellung jedoch vor der Auslieferung Ihres Programmes wieder auf den ursprünglichen Wert zurücksetzen.

Hinweis: Bei einem Dual Mode-Programm werden nur die Segmente im .EXE-Hauptmodul komprimiert. Overlay-Dateien, egal ob interne oder externe, werden nicht komprimiert, wie im nachstehenden Abschnitt ‘Komprimieren von DOS Real Mode-Programmen’ erläutert. Eine umfassende Kompression wird daher eher mit einem Programm für den DOS-Extender als mit einem Dual Mode-Programm erzielt, allerdings auf Kosten der Fähigkeit, im Real Mode abzulaufen - falls benötigt.

.EXE- und .DLL-Dateikomprimierung

Komprimieren von DOS-Real Mode-Programmen

Zur Linkzeit wird eine kleine Dekomprimierungsroutine an den Beginn der .EXE-Datei angefügt und danach wird das gesamte Root (ohne Overlays) als ein einziger Block komprimiert. Zur Dekomprimierung werden etwa 100 KBytes zusätzlicher Speicher zur Ladezeit benötigt, dieser Speicher wird jedoch an das Programm wieder zurückgegeben, bevor dieses gestartet wird.

Beachten Sie, daß im Moment keine Overlays, egal ob intern oder extern, von Blinker komprimiert werden. Hierfür gibt es hauptsächlich zwei Gründe:

- Wären Overlays komprimiert, müßte ein Extra-Speicherbereich, in dem die Overlays dekomprimiert werden, dem Programm zur Laufzeit zur Verfügung gestellt werden. Dies bedeutet jedoch auch eine Reduzierung des verfügbaren Speichers für den Rest des Programmes.
- Jedes Overlay wird separat geladen und ist gewöhnlich nur ein paar KBytes groß, müßte aber für beste Performance separat dekomprimiert werden. Die Kompressionsrate kleiner Blöcke ist jedoch minimal im Vergleich zu der großer Blöcke, also sind die Vorteile entsprechend gering.

Diese Probleme sind jedoch nicht unüberwindlich und Blinker wird laufend weiterentwickelt. Lesen Sie daher bitte die Datei README.WRI auf der Distributionsdiskette zu letzten Informationen bezüglich

Komprimierung und weiterer Eigenschaften.

Die Swap-Funktion des Blinkers

Überblick über die Swap-Funktion

Sichern des derzeit ausgeführten Programms

Ausführen des untergeordneten Programms

Wiederherstellen des übergeordneten Programms

Problembehandlung bei der Swap-Funktion

Die Swap-Funktion des Blinkers

Überblick über die Swap-Funktion

Die integrierte Speicher-Swapfunktion erlaubt Ihnen, aus Ihrem Programm heraus eine neue Shell zu öffnen, um andere Programme, wie eine Textverarbeitung, ein Kopierprogramm oder den DOS-Prompt zu aktivieren. Fast der gesamte Speicher, der von dem übergeordneten Programm belegt wurde, wird freigegeben und steht dem Einsatz des untergeordneten Programms zur Verfügung.

Hier liegt der Unterschied zu konventionellen Befehlen wie exec(), SHELL oder RUN, die die Bibliotheken der Programmiersprache zur Verfügung stellen, da das übergeordnete Programm während der Ausführung des untergeordneten Programms aus dem Speicher entfernt wird.

Die Swap-Funktion wird in folgenden Schritten vollzogen:

- das gegenwärtig ausgeführte Programm (übergeordnetes Programm) wird, bis auf einen kleinen Kern im Speicher, ins EMS, XMS oder auf die Festplatte gesichert
- das untergeordnete Programm wird ausgeführt
- das Abbild des übergeordneten Programms wird wieder zurückgeholt und die Ausführung fortgesetzt

Lesen Sie bitte auch den Abschnitt ‘Ausführung von Protected Mode-Programmen’ in Kapitel 5 zu bestimmten Einzelheiten und Einschränkungen hinsichtlich der Verschachtelung mehrerer DOS-Extender-Programme.

Die Swap-Funktion des Blinkers

Sichern des derzeit ausgeführten Programms

Das Programm-Abbild kann auf eines der drei unterstützten Auslagerungs-Medien gesichert werden - XMS 2.0 (oder höher)-Speicher, angelegt durch Microsoft's HIMEM.SYS-Treiber oder entsprechende EMS 3.2 (oder höher)-Speicher oder eine Swap-Datei auf der Festplatte.

Wenn nötig, nimmt die Swap-Funktion einen Übergang von einem Medium zum anderen vor, indem Teile des Speicherabbildes auf einem Speichermedium gesichert werden, der Rest auf dem anderen. Unter keinen Umständen wird jedoch mehr als eine einzige Swap-Datei erzeugt.

Die Swap-Funktion kann von einem übergeordneten Real Mode- oder Protected Mode-Programm aufgerufen werden und kann ein untergeordnetes Real Mode- oder Protected Mode-Programm aufrufen, es wird jedoch nur der konventionelle Speicherbereich des übergeordneten Programmes ausgelagert. Daher ist es oft wichtig, die Größe des XMS- oder EMS-Speichers, der dem ersten Programm zugewiesen wird, so zu beschränken, daß die Swap-Funktion ausgeführt werden kann und dem untergeordneten Programm noch Speicher zur Verfügung steht.

XMS-/EMS-Speicher

Der Einsatz von EMS- oder XMS-Speicher kann durch folgende zwei Funktionen kontrolliert werden: SWPUSEXMS() und SWPUSEEMS(). Der Einsatz von XMS und EMS wird als Voreinstellung angenommen, das heißt, EMS oder XMS wird benutzt, wenn es verfügbar ist.

Zusätzlich kann die Größe des XMS- oder EMS-Speichers für das untergeordnete Programm durch Verwendung der zwei Funktionen SWPFREXMS() und SWPFREEMS() gesteuert werden.

Festplatten-Swapdatei

Für den Fall, daß kein EMS- oder XMS-Speicher zur Verfügung steht oder die Verwendung nicht zugelassen wurde, wird das Abbild in einer Swap-Datei gesichert. Das Verzeichnis, in dem diese Datei gesichert werden soll, kann als Parameter an die Swap-Funktion übergeben werden. Alternativ kann es auch über die SWAP-Umgebungsvariable spezifiziert werden, die Vorrang gegenüber dem übergebenen Parameter hat. Standard ist das aktuelle Verzeichnis.

Jede Swap-Datei wird mit einem eindeutigen Namen erzeugt. Unter DOS 3.3 oder höher wird DOS veranlaßt, einen eindeutigen Namen zu erzeugen, unter DOS 2.x erstellt die Swap-Funktion selbst einen eindeutigen Namen. Die Image-Datei wird mit den Attributen 'nur lesen' und 'versteckt' erzeugt, um eine unbeabsichtigte Löschung während der Ausführung des 'untergeordnetes Programmes' zu vermeiden.

Abspeichern der Systemeinstellungen

Die Swap-Funktion kann folgende Einstellungen sichern und wiederherstellen:

- erweiterte Datei-Handle - Tabelle
- Interrupt-Vektorentabelle
- Video-Modus und Seite (optional)
- aktuelles Verzeichnis (optional)
- EMS-Pageframe Mapping-Status
- Maus-Ereignisroutinen und Aufrufmaske

Der residente Kern

Eine kleine Menge Code und Daten, residenter Kern genannt, verbleibt während der Auslagerung immer im Speicher. Dieser Code hat verschiedene Funktionen, die wichtigste ist, das übergeordnete Programm wieder herzustellen, wenn das untergeordnete Programm beendet wurde.

Momentan gibt es zwei mögliche Stellen, wo der residente Kern abgelegt sein kann - am Anfang des konventionellen Speichers oder in einem Upper Memory Block (UMB).

UMBs (auch als LOADHI-Bereich bekannt) sind Speicherbereiche zwischen den Adressen 640 KB und 1 MB. Sie befinden sich außerhalb des konventionellen Speichers, aber innerhalb des 1-MB-Real Mode-Adressbereiches der Systeme mit 80x86-Prozessoren.

UMBs sind normalerweise nur auf Rechnern mit 386er Prozessoren (oder höher) verfügbar. Diese können durch Speicher-Manager, wie EMM386 in Verbindung mit DOS 5.0, QEMM von QuarterDeck oder 386MAX von Qualitas, zur Verfügung gestellt werden.

Die Swap-Funktion benutzt standardmäßig den UMB, wenn verfügbar. Dadurch erhöht sich die Menge des verfügbaren Speichers für das untergeordnete Programm um einige KB. Der Einsatz der UMBs durch die Swap-Funktion kann durch die SWPUSEUMB()-Funktion kontrolliert werden.

Größe des residenten Kerns

Unter normalen Umständen liegt die Größe des residenten Kerns bei 7-8 KB. Wenn genügend UMB verfügbar ist, umfaßt der Overhead im konventionellen Speicher lediglich die Größe der Umgebungsvariablen des übergeordneten Programms plus der Größe des Befehls-Interpreters. Ist das übergeordnete Programm ungewöhnlich klein, kein UMB verfügbar und der Kern nicht in der Lage, sich selbst auszulagern, dann kann der Overhead auch schon einmal bis zu 11 KB groß werden.

Die Swap-Funktion des Blinkers

Ausführen des untergeordneten Programms

Nachdem das übergeordnete Programm gesichert und der residente Kern ausgelagert wurde, wird der gesamte Speicher, der durch das übergeordnete Programm angefordert wurde, freigegeben und das untergeordnete Programm wird, wenn möglich, am Anfang des DOS-Speichers ausgeführt.

Die Swap-Funktion ruft immer eine Kopie des aktuellen Befehls-Interpreters auf, der den angeforderten DOS-Befehl oder ein Programm ausführt. Dies ermöglicht es, alle vom DOS-Prompt aus ausführbaren Befehle, Batch-Dateien oder Programme auch über die Swap-Funktion auszuführen.

Ist das untergeordnete Programm oder ein Befehl ausgeführt worden, übernimmt der residente Kern wieder die Steuerung.

Die Swap-Funktion des Blinkers

Wiederherstellen des übergeordneten Programms

Nach Beendigung des untergeordneten Programmes lädt der residente Kern das exakte Speicher-Abbild des übergeordneten Programmes wieder ein, so wie es vor Ausführung des untergeordneten Programmes war. Jeder dafür benutzte XMS-/EMS- und/oder Plattenspeicher wird freigegeben und die Ausführung des übergeordneten Programmes fortgesetzt.

Ereignisgesteuerte Routinen

Sollte Ihr Programm Interrupt-Handler beinhalten, entweder von Ihnen erstellt oder aus einer Funktions-Bibliothek eingelinkt, dann werden die Interrupt-Vektoren automatisch von der Swap-Funktion korrekt gesichert und wieder hergestellt.

Maus

Wenn in Ihrem Programm jedoch eine Maus (oder ähnliches) eingesetzt wird, dann wird der Maus-Status möglicherweise nicht korrekt restauriert, wenn die Maus während des Swap-Vorganges bewegt oder zurückgesetzt wurde.

Aus diesem Grund sollten Sie folgendes tun:

- Sichern des Maus-Status
- Abschalten der Maus-Funktion
- Starten der Swap-Funktion
- Einschalten der Maus-Funktion
- Restaurieren des Maus-Status

Es ist besonders wichtig, daß Sie die Maus vor Ausführung der Swap-Funktion sperren, falls Sie eine alternative Mausbehandlungs-Funktion zusammen mit dem Maustreiber eingebunden haben, anderenfalls wird der Maustreiber versuchen, Ihre Routine zu starten, wenn eine Mausbewegung stattfindet. Das Ergebnis wird dann meistens ein Programmabsturz sein, da sich Ihr Programm zu diesem Zeitpunkt schon nicht mehr im Speicher befindet.

Standardmäßig wird von der Swap-Funktion nur die aktuelle Adresse der Maus-Behandlungsroutine und die Aufrufmaske, jedoch keine weiteren Informationen gespeichert und wiederhergestellt.

Callback-Funktionen

Eine Callback-Funktion ist in diesem Zusammenhang eine Routine, deren Adresse jedem externen Programm zugeführt wird, welches Ihre Funktion evtl. periodisch aufruft - der oben erwähnte alternative Maus-Handler ist ein Beispiel für diesen Mechanismus.

Sollte Ihre Applikation eine ‘Callback-Routine’ installiert haben, muß die Routine ausgeklinkt werden, bevor die Swap-Funktion ausgeführt wird, und eingeklinkt werden, nachdem das ‘übergeordnetes Programm’ restauriert wurde.

Speicherresidente Programme (TSR)

Wenn die Swap-Funktion das Abbild des übergeordneten Programms restauriert, wird jedes während des Swap-Vorganges installierte TSR aus dem Speicher entfernt. Das mag zwar mit vielen TSR-Programmen

korrekt funktionieren, jedoch ist es sicherer (und empfehlenswert), daß die TSR-Programme unter Ausnutzung ihrer Removal-Funktion vor Rückkehr in das übergeordnete Programm aus dem Speicher entfernt werden.

Die Swap-Funktion des Blinkers

Problembehandlung bei der Swap-Funktion

Das am häufigsten vorkommende Problem beim Einsatz der Swap-Funktion besteht darin, daß die COMSPEC-Umgebungsvariable nicht auf den gültigen DOS-Kommandoprozessor zeigt. Überprüfen Sie also, daß COMSPEC auf den richtigen COMMAND.COM für die DOS- bzw. Windows-Version, die gerade läuft, zeigt.

Wenn dieser Fehler nicht vorliegt, esteht der nächste Schritt darin, die Funktionen SWPERRMAJ() und SWPERRMIN() zur Identifizierung der Fehlerursache einzusetzen. Bitte lesen Sie die genauere Beschreibung zum Einsatz dieser Funktionen in der Online-Hilfe.

Wenn dies Funktionen keinen aussagefähigen Wert zurückgeben oder die Lösung, die zu der Fehlernummer vorgeschlagen wird, entweder nicht in Frage kommt oder nicht hilft, ist der nächste Schritt, zu versuchen, den DOS-Befehlsprompt als Kindprozess aufzurufen, indem COMMAND.COM als Kindprozess angegeben wird.

Wenn sogar Swap zum COMMAND.COM mit dem ursprünglichen übergeordneten Programm nicht gelingt, dann sollten Sie ein kleines Testprogramm schreiben, das nichts anderes macht als den Kindprozess aufzurufen und beliebige Fehlernachrichten nach der Rückkehr vom untergeordneten Prozess auszugeben. Wenn dies wiederum fehlschlägt, dann versuchen Sie von diesem Testprogramm aus, zum DOS-Prompt wie oben beschrieben zu swappen..

Bitte lesen Sie auch den Abschnitt ‘Ausführen eines Protected Mode-Programmen’ in Kapitel 3. Dort finden Sie weitere Informationen und auch Hinweise auf Einschränkungen bzgl des mehrmaligen Verschachtelns von DOS-Extender-Programmen. Ein besonderes Beispiel dazu ist ein DOS-Extender-Programm, das in einer DOS-Box von Windows läuft und dort PKZIP startet, was wiederum ein DOS-Extender-Programm ist. Man muß bei PKZIP den Kommandozeilenparameter "-)" angeben, wenn PKZIP kein DPMI nutzen soll..

Allgemeine Debugger-Unterstützung

Überblick zur Debugger-Unterstützung

Erzeugen von Debugger-Informationen

Selektive Debugger-Informationen

NuMega SoftICE/W

Periscope und Periscope/32

Microsoft CodeView

Borland Turbo Debugger

Symantec SCD

CA-Clipper Debugger

Blinker Umgebungseinstellung /DI3

Allgemeine Debugger-Unterstützung

Überblick zur Debugger-Unterstützung

Blinker kann CodeView-Format-.EXE-Dateien mit der gesamten Debugger-Information, die am Dateiende abgelegt ist, optional erzeugen. Die Versionen 0 bis 5 der CodeView-Formate werden unterstützt und die speziell zu schreibende Version kann zur Linkzeit zusammen mit den Teilen des Programmes, die zu debuggen sind, spezifiziert werden. .EXE-Dateien ohne Overlays sollten auch voll kompatibel zu anderen Debuggern sein, die das CodeView-Format lesen können, wie zum Beispiel Periscope und Soft-Ice.

Blinker unterstützt eine Reihe von Quelltext-Debuggern für DOS und Windows, entweder direkt durch CodeView-Information oder indirekt durch eine Map-Datei oder durch Umwandlungs-Utilities, die mit diesen Debuggern ausgeliefert werden.

Einige Debugger benötigen für das Debuggen exklusiven Zugriff auf die ausführbare Datei, was zu Problemen bei Programmen mit Overlays führen kann - das übliche Symptom davon ist der Blinker-Fehler 1203. Sie können dieses Problem umgehen, indem Sie das Blinker Linkskriptkommando READONLY einsetzen, um die ausführbare Datei mit dem Read Only-Attribut zu erzeugen, oder indem Sie über den DOS-Befehl ATTRIB dieses Read Only-Attribut nach dem Linkvorgang manuell setzen.

Allgemeine Debugger-Unterstützung

Erzeugen von Debugger-Informationen

CodeView Debug-Informationen müssen von den .OBJ-Dateien geliefert werden, die vom eingesetzten Compiler oder Assembler generiert wurden. Daher ist die erforderliche Information genauer als die normalerweise in einer .OBJ-Datei enthaltene.

Die meisten Compiler haben zumindest einen Debug-Schalter, um diese Zusatzinformation zugänglich zu machen und zumindest eine Option, um Quellcode-Zeilennummern zu generieren. Jedes Modul, das mit dem Debugger untersucht werden soll, muß mit den entsprechenden Debugger-Schaltern des jeweiligen Compilers kompiliert werden. Bitte schlagen Sie in Ihrem Compiler-Handbuch für weitere Informationen nach, falls Ihr Compiler nachstehend nicht aufgeführt ist:

Borland Assembler	-Zi
Borland Compiler	-v
FORCE	-z
Microsoft Assembler	-Zi
Microsoft Compiler	-Zi
Symantec C++	-g
WATCOM C++	-d2

Um Blinker mitzuteilen, daß die CodeView-Information in die .EXE-Datei eingefügt werden soll, geben Sie einfach den CODEVIEW-Befehl auf der Kommandozeile oder an einer beliebigen Stelle der Skriptdatei ein.

Der MAP-Befehl wird mit den Parametern S und A eingesetzt, um eine Mapdatei zu erzeugen, die alle Segmente und alle Public-Symbole in Adressenreihenfolge auflistet. Die Mapdatei ist kompatibel zu Microsoft Link, so daß die meisten anderen Debugger in der Lage sind, die Mapdatei für eigene Debug-Zwecke zu nutzen.

CodeView-Versionen

Die hier aufgeführten Versionsnummern von CodeView beziehen sich auf die Versionsnummer der Information, nicht auf eine spezielle Ausgabe des Debugger-Programmes CodeView.

Die von Blinker produzierte Information entspricht dem jüngsten Support-Level: CodeView Version 5. Manche Debugger benötigen frühere Versionen von Informationen - hier müssen Sie Blinker instruieren, daß er Informationen entsprechend dem früheren Level generiert. Zu diesem Zweck steht der Befehl CVVERSION zur Verfügung, der beliebig auf der Befehlszeile oder in der Skriptdatei angegeben werden kann.

```
CVVERSION 2      # Biete Level 2 Codeview-Unterstützung
              # wie für Microsoft C 5.1 and 6.0 benötigt
```

Gültige Werte sind 0, 1, 2, und 5. Geben Sie keine Versionsnummer für CodeView-Informationsunterstützung an, die höher ist, als es Ihr Debugger unterstützt.

Die Informations-Versionsnummer ist vorgabemäßig auf 5 eingestellt, da dieser Wert von Microsoft C++ 7.0 als auch von Visual C++ benötigt wird. Am besten geben Sie den höchsten Wert an, den Ihre Version von CodeView unterstützt, da in den höheren Versionen mehr Funktionalität ergänzt wurde. Sehen Sie hierzu den Befehl CVVERSION in Kapitel 9 zu einer Aufstellung an Compilern und deren benötigten Versionsnummern.

Allgemeine Debugger-Unterstützung

Selektive Debugger-Informationen

Wenn alle Module mit Debugger-Information kompiliert werden, was häufig vorkommt, können, anstatt daß riesige Mengen von Debugger-Informationen erzeugt werden, die CVON- und CVOFF-Befehle eingesetzt werden, um die Bereiche zu begrenzen, für die CodeView-Informationen abgelegt werden.

Der CVON-Befehl zeigt an, daß alle Dateien, die bis zum nächsten CVOFF-Befehl in der Skriptdatei folgen, CodeView-Informationen tragen, die in die .EXE-Datei mit eingebunden sein sollten. Diese Befehle können paarweise eingesetzt werden, um gezielt CodeView-Informationen für Teile des Programmes zu ermöglichen.

Sie sollten anstatt des globalen CODEVIEW-Befehles eingesetzt werden, jedoch kann der CVVERSION <version>-Befehl zur Einstellung der CodeView-Version in jedem Falle benutzt werden.

Beispiel:

```
# TEST.LNK mit selektivem Debuggen
FILE start       # Debuggen nicht möglich
CVON
FILE one      # Debuggen möglich
FILE two      # Debuggen möglich
FILE three       # Debuggen möglich
CVOFF
FILE four        # Debuggen nicht möglich
FILE five        # Debuggen nicht möglich
FILE six         # Debuggen nicht möglich
FILE seven       # Debuggen nicht möglich
FILE eight       # Debuggen nicht möglich
CVON
FILE nine     # Debuggen möglich
CVOFF
LIB inhouse      # Debuggen nicht möglich
```

Overlays

Für ein Programm mit Overlays muß während der Fehlersuche unter CodeView mit Hilfe des Befehls SECTION INTO die Overlay-Methode external verwendet werde. Externes Overlaying stellt eine globale Option dar, d.h., sämtlicher Code muß entweder intern oder extern sein.

Um zum Beispiel das Modul TEST2 aus dem obigen Programm in Microsoft C++ unter dem CodeView-Debuggen testen zu können, kompilieren Sie es mit Debug-Informationen:

```
CL -c -Zi test2.c
```

Modifizieren Sie die Skriptdatei durch Verlegen von TEST2 in das Hauptverzeichnis und fügen Sie den CODEVIEW-Befehl hinzu. Achtung: Blinker benötigt externe Overlays, um CodeView-Informationen für ein Programm erzeugen zu können

```
#TEST.LNK mit kompletten Debugging

CODEVIEW           # beinhaltet CodeView-Information
CVVERSION 5
SECTION INTO TEST  # Externes Overlay erforderlich
BEGIN              # Beginn des Overlaybereiches
  FILE TEST1       # Erste Overlaydatei
  FILE TEST3       # Dritte Overlaydatei
```

```
END                 # Ende des Overlaybereiches
FILE TEST2          # Modul, das debuggt werden
                    # soll, befindet sich nun im Root
```

Linken und starten Sie das Programmes unter CodeView erneut:

```
BLINKER @TEST
CV TEST
```

Allgemeine Debugger-Unterstützung

NuMega Soft-Ice/W

Blinker enthält volle Unterstützung für den Soft-Ice/W Debugger (von NuMega Technologies) für Real, Protected, Dual und Windows Programme auf einem EINZIGEN RECHNER. Soft-Ice/W lädt tatsächlich Windows und sitzt auf der untersten Systemebene, so daß es alles auf Quellcode-Ebene debuggen kann, einschließlich VxDs und DOS Protected Mode Programme.

Um Soft-Ice/W mit DOS Real Mode oder Protected Mode Programmen einzusetzen, kompilieren Sie die geeigneten Module so, daß Debuginformationen eingeschlossen werden und linken dann mit dem CODEVIEW Linkskript-Befehl. Wenn die EXE mit den Debuginformationen erstellt ist, verwenden Sie den geeigneten SoftICE/W Symbol-Lader (hängt von der SoftICE/W-Version ab), um die Debugsymbole in SoftICE/W zu laden.

Der Symbol-Lader bei SoftICE/W für Windows 95/NT heiß NMSYM. Um also die Symbole zu laden, starten Sie NMSYM wie folgt:

NMSYM /TRANSLATE /LOAD:SYMBOLS <.EXE- oder .DLL-Name>

Der Symbol-Lader bei SoftICE/W für Windows 3.1 heißt WLDR.EXE. Um also die Symbole zu laden, starten Sie WLDR wie folgt:

WLDR <.EXE-Name>.SYM

Eine .SYM-Datei wird automatisch erstellt. Verwenden Sie WLDR <.EXE-Name> nicht ohne die Angabe einer Erweiterung, da DOS-Extender- Programme zur Zeit nicht zur gleichen Zeit wie die Symbole mit WLDR geladen werden können.

Damit die Unterbrechung durch den INT 3 Debugger-Interrupt von SoftIce/W erkannt wird, verwenden Sie den Befehl 'I3HERE ON' beim Aufruf von Soft-Ice/W (beim Start von Soft-Ice/W mit Ctrl D), oder geben Sie ihn in der INIT-Zeile in der Soft-Ice/W Initialisierungsdatei WINICE.DAT an. To add an INT 3 instruction to a C/C++ program simply use the directive

__ASM INT 3

Dann schaltet sich Soft-Ice/W ein, wenn es auf ein INT 3 im Code stößt, wenn Control D gedrückt wird, oder es kann direkt beim Start des Programms eingeschaltet werden, wenn Sie den Wert /DI3 in der Umgebungsvariablen BLINKER setzen, wie im vorigen Abschnitt beschrieben. Soft-Ice/W schaltet sich dann bei einem far JMP auf die allererste Zeile des eigentlichen Programms ein, auf die Sie mit F8 steppen können. Wie beereits gesagt, bringt Sie bei Hochsprachen der Befehl 'G MAIN' an den Beginn des Hauptprogramms.

Wenn Sie den SoftICE/W –Lader verwenden, um die tatsächliche DOS-Extender-EXE zu laden, ist der erste angezeigte Code der Blinker DOS-Extender, der immer noch im DOS Real Mode läuft. Sie können

den Code des DOS-Extender s überspringen, indem Sie die RETF-Anweisung in der Nähe von Offset 0F0h im angezeigten Code-Segment suchen und dann einen G 0F0-Befehl ausführen. Nach der RETF-Anweisung schaltet der DOS-Extender in den Protected Mode um und kehrt dann zurück, um die Kernel-Initialisierung von Blinker an der Startposition des eigentlichen Programms aufzurufen.

Beim Debuggen von DOS-Extender-Prohgrammen sind die Segmente noch nicht verfügbar, wenn auf sie zum ersten Mal zugegriffen wird. Das führt dazu, daß SoftICE oft in die Behandlungsroutinen von Windows oder dem Blinker-Extender für nicht verfügbar (‘not present’) verzweigt, was etwas verwirrend sein kann. Erzwingt man beim Start, daß alle Segmente im Vorhinein geladen werden, wird dieses Verhalten verhindert. Dies sollte jedoch nur zu Debugzwecken eingesetzt werden und nicht für eine Auslieferungsversion. Mit dem folgenden Vierzeiler am Ende der Blinker LNK-Skriptdatei erzwingen Sie, daß alle Segmente (CODE and DATA) im Vorhinein geladen werden:

```
DEFBEGIN
        CODE PRELOAD
        DATA PRELOAD
DEFEND
```

Neues Linken nicht vergessen! Weitere detaillierte Erklärung der Funktionsweise dieser Befehle finden Sie in den entsprechenden Beschreibungen in der Online-Hilfe.

Anmerkungen zu SoftICE für Windows 95

Da Windows bezeichnenderweise nur mit einer LDT (Local Descriptor Table) läuft, während DOS-Extender-Programme beim Start immer ihre eigene LDT allozieren, gibt es zwei Szenarien, auf die Sie achten oder die Sie vermeiden sollten.

Das erste tritt auf, wenn Soft-Ice/W über Ctrl-D aufgerufen wird, während man außerhalb des DOS-Extender-Programmes ist. Die aktive LDT gehört zu Windows und nicht zum Programm, d.h. alle Selektoren beziehen sich auf Windows-Code und -Daten, nicht auf die des Programmes. Das Ergebnis ist, daß alle Symboladressen des Programmes inkorrekt und ungültig sind, sie dürfen nicht für Aktionen wie ‘Speicher anzeigen oder modifizieren’ oder ‘Breakpoints setzen’ etc. verwendet werden.

Das zweite tritt auf, wenn ein Breakpoint im DOS-Extender-Programm gesetzt ist und dann über Alt-ESC zurück in Windows geschaltet und ein anderes Programm ausgeführt wird. In diesem Fall ist es so, daß Soft-Ice/W den Breakpoint auf die Windows 95-LDT umstellt, statt sie auf der Extender-LDT zu belassen, wodurch der Breakpoint irgendwo anders innerhalb Windows ausgelöst wird.

Außer diesen beiden Fallstricken unter Windows 95, kann Soft-Ice/W alle Typen von mit Blinker gelinkten Programmen auf einem einzelnen Rechner debuggen. Lesen Sie bitte die neuesten Hinweise in der Datei DEBUG.WRI im Blinker-Verzeichnis.

Anmerkungen zu Borland C/C++

Borland C/C++ verwendet ein eigenes Format für seine Symbol-Debuginformationen. Deshalb unterstützt Blinker momentan nur die Quelltextzeilennummerinformation und globale Variablen bei Borland C/C++.

Vollständige Unterstützung der Debuginformationen bei Borland C/C++ ist für eine spätere Blinker-Version vorgesehen.

Allgemeine Debugger-Unterstützung

Periscope und Periscope/32

Blinker bietet volle Unterstützung des Periscope-Debuggers für Real Mode-Anwendungen und des Periscope/32-Debuggers für Real, Protected und Dual Mode-Programmen wie auch für Windows-Programme unter Einsatz von zwei Computern. Periscope/32 benötigt einen Rechner zur Ausführung des zu debuggenden Programmes und einen zweiten, auf dem die Quelltexte abgelegt sind, um den Debugger selbst auszuführen.

Um Periscope mit DOS Real Mode- oder Protected Mode-Programmen einzusetzen, linken Sie einfach für CODEVIEW mit CVV 2 oder 5, rufen TS /e xxxx auf und starten danach das Programm xxxx.EXE direkt oder über RUN xxxx.EXE.

Allgemeine Debugger-Unterstützung

Microsoft CodeView

Hinweis: Der Microsoft CodeView-Debugger KANN NICHT zum Debuggen von DOS-Extender-Programmen eingesetzt werden. Verwenden Sie hierfür einen Debugger wie etwa Soft-Ice/W oder Periscope/32 für Windows.

Blinker unterstützt in CodeView das Debuggen von REAL Mode-Programmen mit Compilern, die CodeView-Informationen der Stufe 0, 1, 2, und 5 erzeugen. Dies umfaßt alle MS-Compiler bis hin zu C/C++ Version 7.x und Visual C++ (C8), Visual BASIC für DOS 1.0, BASIC PDS 7.1, Pascal 4.0, Fortran 5.0 und MASM 6.x. Zum Zeitpunkt der Erstellung dieses Handbuches erfüllen auch Compiler von anderen Herstellern, die CodeView-Informationen generieren, diese Anforderungen.

Overlays und CodeView Versionen 1 - 3

Damit nun auch Kompatibilität zu früheren Versionen von CodeView gegeben ist, trotzdem aber die Vorteile von dynamischen Overlays beibehalten werden, erzeugt Blinker einen statischen Overlay-Bereich im .EXE-Hauptprogramm, dessen Größe durch das größte überlagerte Objektmodul bestimmt wird. Bei der Programmausführung identifiziert der Overlaymanager jene Module, die sich in einem ‘CodeView-Bereich’ befinden und kopiert sie in den statischen Bereich zur Ausführung. Das Anlegen des statischen Overlay-Bereichs innerhalb des Programmes erhöht die Ladegröße des Programmes um den Wert des größten überlagerten Moduls, für das CodeView-Informationen vorhanden und aktiviert sind.

Um nun den Einzelschrittmodus zu ermöglichen und auch in überlagerten Modulen Fehler zu verfolgen, erzeugt Blinker eine Datei mit dem gleichen Namen wie die ausführbare Datei, jedoch mit der Erweiterung .VEC. Diese Datei muß CodeView zur Laufzeit zur Verfügung gestellt werden, so als sei sie eine Ihrer Quelltext-Dateien. Bei der Fehlerverfolgung in Ihrem Programm folgen Sie nun den Anweisungen, die in dieser Datei gegeben sind: zu Einzelschritten wohin und wann, zu Overlay-Vektoren und zum Overlay-Manager.

Das Debuggen von überlagertem Code unter älteren Versionen von CodeView bringt eine Einbuße an Geschwindigkeit mit sich, die in erster Linie daran liegt, daß der Overlaymanager Module in den statischen Overlaybereich kopieren muß. Effektiv bedeutet dies eine ‘Worst Case’-Overlay-Situation, in

der Module konstant in und aus einem einzelnen Speicherbereich ein- bzw. ausgelagert werden. Die Auswirkung hängt von dem Programm (und seinen Modulen) ab, die mit dem Debugger untersucht werden. Es gibt zwei Dinge, die Sie unternehmen können, um dieses Problem zu verringern:

- Beschränken Sie die Codegröße, die untersucht werden soll durch Verwendung von CodeView-Bereichen
- Erhöhen Sie die Größe des Overlay-Cache, um Overlay-Operationen zu beschleunigen

Allgemeine Debugger-Unterstützung

Borland Turbo Debugger

Der Borland Turbo Debugger kann mit durch Blinker erstellten .EXE-Dateien arbeiten, wenn das von Borland gelieferte Hilfsprogramm TDCONVRT verwendet wird. Dieses Hilfsprogramm konvertiert eine CodeView-Format-.EXE-Datei in das Turbo Debugger-Format. Für weitere Informationen über dieses Utility schlagen Sie bitte in der entsprechenden Borland-Dokumentation nach.

Aktuelle Informationen zu Blinker's Turbo Debugger-Unterstützung entnehmen Sie bitte dem Dokument DEBUG.WRI auf der Originaldiskette. Jede dort enthaltene Information ersetzt die jeweilige in diesem Abschnitt.

Wenn es keine weiteren Information für den Gebrauch des Turbo Debuggers gibt, erzeugen Sie die CodeView-Format-.EXE-Datei wie im vorherigen Abschnitt beschrieben, um sie danach unter Einsatz der TDCONVRT.EXE in das Turbo Debugger Format umzuwandeln:

```
BLINKER @TEST
TDCONVRT TEST
TD TEST
```

Allgemeine Debugger-Unterstützung

Symantec SCD

Symantec C++ 4.0 und höher unterstützt Code View 4.x und seine CodeView-Informationen direkt. Wenn diese Compiler verwendet werden, dann können Sie einfach CVVERSION auf 5 zu setzen und wie gewohnt debuggen. Wenn Sie den Fehler 1138 ausgegeben bekommen, 'kann Programm nicht finden und ausführen,' dann sollten Sie das CodeView-Pack-Hilfsprogramm von Symantec in CVPACK.EXE kopieren.

Allgemeine Debugger-Unterstützung

CA-Clipper Debugger

Mit Blinker erzeugte CA-Clipper Sommer ’87- oder 5.x-Programme sind voll kompatibel zu Debuggern dieser Versionen. Bitte schlagen Sie für weitere Informationen in der Abhandlung zur entsprechenden CA-Clipper-Version in Kapitel 8 und in der CA-Clipper-Dokumentation nach.

Allgemeine Debugger-Unterstützung

Blinker Umgebungseinstellung /DI3

Die Einstellung /DI3 (DEBUG INT 3) in der Umgebungsvariablen BLINKER verursacht einen Debug-Interrupt 3 direkt vor dem far-Sprung auf das erste Byte des eigentlichen Programms. Das ist nachdem der Extender und der Blinker-Kern geladen und initialisiert sind. Über diese Anweisung können Sie mit normalem Schritt auf das erste Byte des eigentlichen Programms gelangen.

In Hochsprachenprogrammen ist dies der Startup-Code für die Bibliothek. Um den Startup der Bibliothek zu überspringen und Ihr Programm (Function MAIN) zu starten geben Sie 'G MAIN' oder etwas Entsprechendes ein.

Debuggen von Real Mode Overlay-Programmen

Es können fast alle CodeView-kompatible Debugger zur Fehlersuche von Real Mode-Programmen ohne Overlays verwendet werden und viele von ihnen eignen sich auch für Programme mit Overlays. In den Produkt-Abschnitten weiter oben und den individuellen Produktbeschreibungen finden Sie weitere Details zu speziellen Debuggern.

Programm-Probleme treten manchmal nur dann auf, wenn einzelne Teile des Codes ins Overlay gelegt oder in einer bestimmten Reihenfolge aufgerufen werden. Aus diesem Grunde wurde der Skript-Befehl MURPHY zur Verfügung gestellt, um zu versuchen, die Probleme zur Debug-Zeit auftreten zu lassen. Der MURPHY-Befehl zwingt den Overlay-Manager, sicherzustellen, daß sich zu jedem Zeitpunkt immer nur ein aktives Overlay im Speicher befindet.

Dieses ‘worst case'-Szenario zwingt alle Aufrufe anderer Overlays, über den Overlay-Manager zu gehen, anstatt daß die Kontrolle direkt an eine nahe Speicheradresse übergeben wird. Dies wiederum sollte alle Probleme, welche durch nicht overlayfähigen Code und nicht unterstützte Referenzen zwischen zwei Overlays entstehen, zum Vorschein bringen.

Der Befehl MURPHY kann überall plaziert werden, sowohl auf der Kommandozeile als auch in der Skriptdatei, da er global gültig ist.

Hier wird ein kleines Testprogramm gelinkt, daß Overlays im 'worst case' testet:

```
# TEST.LNK
```

```
BEGIN                # Beginn des Overlaybereiches
  FILE TEST1         # Erste Overlay-Datei
  FILE TEST2         # Zweite Overlay-Datei
  FILE TEST3         # Dritte Overlay-Datei
END                  # Ende des Overlay-Bereiches
MURPHY               # Einschalten 'worst case'- Overlaying
BLI CACHE XMS 256    # Cache für Overlays zur Erhöhung
                   # der Geschwindigkeit
BLI CACHE XMS 256    # Cache für Overlays zur Erhöhung
                   # der Geschwindigkeit
```

Die Endversion einer Anwendung sollte NICHT mit der MURPHY-Option ausgeliefert werden, da sie langsamer abläuft als normal und es möglicherweise zu Fehlern kommt, die unter normalen Umständen nicht auftreten.

Unterstützung für Profiler

Microsoft Profiler

Der Microsoft Profiler unterstützt im Moment keine Overlayprogramme, jedoch sind .EXE-Dateien ohne Overlays, die mit Blinker in Verbindung mit CodeView-Informationen erzeugt wurden, voll kompatibel zum Microsoft Profiler.

Erzeugen Sie eine .EXE-Datei ohne Overlays für die Verwendung mit CodeView wie im obigen Abschnitt beschrieben und nutzen Sie sie dann zusammen mit dem Profiler genauso, als hätten Sie sie mit Microsoft LINK erstellt.

Um das oben erzeugte Programm TEST mit dem Profiler zu bearbeiten und die Anzahl der Durchläufe der aufgerufenen Prozeduren aufzuzeichnen, können Sie zum Beispiel die FCOUNT-Batchdatei nutzen, die zusammen mit dem Microsoft Profiler zur Verfügung gestellt wird:

```
FCOUNT TEST
```

Die Ausgabe des Profilers erfolgt in die Datei TEST.OUT und kann mit Hilfe eines ASCII-Texteditors bearbeitet werden.

Gezieltes Profiling kann durch den Einsatz der CVON- und CVOFF-Befehle vorgenommen werden, um lediglich CodeView-Informationen für die mit dem Profiler zu bearbeitenden Dateien aufzunehmen, wie im zweiten Beispiel oben beschrieben.

Turbo Profiler

Ebenso wie der Turbo Debugger mit durch Blinker erzeugten und mit CodeView-Informationen versehenen .EXE-Dateien, welche durch TDCONVRT konvertiert wurden, arbeiten kann, kann dies auch der Turbo Profiler.

Die .EXE-Datei wird, genau wie im Abschnitt oben für den Turbo Debugger beschrieben, vorbereitet und dann unter dem Profiler geladen.

Bitte entnehmen Sie weitere Einzelheiten der Turbo Profiler-Dokumentation.

Erzeugen von Demo-Versionen

Blinker ermöglicht dem Entwickler zur Linkzeit die Erstellung einer Demo-Version seiner DOS- oder DOS-Extender-Anwendung. Dies wird durch die zur Verfügung gestellten Befehle zur Beschränkung auf eine vorgegebene Nutzungszeit in Minuten oder bis zu einem Grenzdatum (‘Verfallsdatum’) oder eine Kombination beider Varianten ermöglicht. Wenn eine dieser Begrenzungen überschritten wird, stoppt der Ablauf des Programmes und eine Nachricht zur Beendigung des Demos wird durch den Blinker Error Handler angezeigt.

Der vorgegebene Blinker Error Handler wird automatisch von Blinker mit der Applikation gelinkt. Dieser wird dann aufgerufen, wenn im Blinker Overlay-Manager ein Fehler auftritt oder eine der zur Linkzeit gesetzten Begrenzungen der Demo-Version überschritten wird. Der Quelltext des Error Handlers steht Ihnen zur Verfügung, sofern Sie die Beendigungsnachricht oder das Ergebnis einer Beendigungsroutine modifizieren wollen. Vergleichen Sie den entsprechenden früheren Abschnitt über den Error Handler in diesem Kapitel.

Beachten Sie: Für Real Mode-Programme werden die Demonstrationsrestriktionen vom Blinker Overlay-Manager nur geprüft, wenn Overlays ausgeführt werden, da Blinker keine Kontrolle über Prozeduren hat, die im Root abgelegt sind. Wenn daher eine Real Mode-Anwendung durch diese Maßnahmen begrenzt wurde, muß zumindest eine Prozedur oder Funktion, die regelmäßig aufgerufen wird, ins Overlay gelegt werden.

Tips zur Verwendung der Demo-Features

- Soll in ein inkrementell unter CA-Clipper gelinktes Programm ein Demo-Feature eingebaut oder verändert werden, dann muß das Programm völlig neu gelinkt werden. Dabei muß entweder das inkrementelle Linken ausgeschaltet sein oder die BIF- oder EXE-Datei des Programms gelöscht werden.
- In CA-Clipper-Programmen muß Nicht-CA-Clipper-Code, zum Beispiel die EXTEND.LIB, in Overlays abgelegt werden, damit sich die Demo-Features auswirken können.
- Sehr kleine .OBJ-Dateien im Overlay-Bereich können durch den Blinker in den konventionellen Speicher verschoben werden, wenn sie kleiner sind als der über BLINKER OVERLAY THRESHOLD angegebene Standard. Achten Sie daher darauf, daß mindestens eine in einem Overlay abzulegende .OBJ-Datei größer als dieses Minimum ist.

Datumsbeschränkung in einem Demo-Programm

Eine Applikation kann so eingeschränkt werden, daß sie nach einem bestimmten Datum nicht mehr lauffähig ist. Das Programm vergleicht das angegebene Ablaufdatum mit dem Systemdatum und wenn ersteres größer als oder gleich dem zweiten ist, wird der Error Handler BLERRPRG aufgerufen, der dann die Applikation abbricht.

Um das Ablaufdatum für eine Applikation zu setzen, nehmen Sie den Befehl

```
BLINKER DEMONSTRATION DATE <JJJJ/MM/TT>
```

in der Skriptdatei auf. Wobei MM den Monat, TT den Tag und JJJJ das Jahr wiedergibt. Nach diesem Datum wird das Programm nicht mehr ablauffähig sein.

Zeitbeschränkung in einem Demo-Programm

Eine Applikation kann so eingeschränkt werden, daß der Overlay-Manager, nachdem er für eine

bestimmte Zeit geladen wurde, die Fehlerbehandlungs-Prozedur BLERRPRG aufruft, welche die Ausführung der Applikation beendet. Das Programm kann erneut gestartet werden und zwar so oft wie gewünscht, aber wiederum nur für die begrenzte Zeit. Nach jedem Ablauf wird der Zeitzähler zurückgesetzt.

Um die Ausführungszeit für ein Programm zu bestimmen, schreiben Sie

```
BLINKER DEMONSTRATION MINUTES <nGrenze>
```

in die Skriptdatei. Der Parameter <nGrenze> wird im Dezimalformat spezifiziert und kann bis 65535 reichen (größere Werte werden ignoriert).

Seriennummern-Verschlüsselung

Mit Blinker kann der Entwickler einen eindeutigen String in die ausführbare Datei einbrennen. Dieser String kann eine Seriennummer, Copyright-Information o.ä. enthalten. Er kann bis zu 100 Zeichen lang sein und wird in der ausführbaren .EXE-Datei in verschlüsselter Form abgelegt.

Der Blinker-Befehl für diesen Zusatz lautet BLINKER EXECUTABLE SERIAL, gefolgt von dem zu verschlüsselnden String. Er kann zur Laufzeit mit der Blinker-Funktion BLISERNUM() (oder BLISERNUMBAS() im Falle von BASIC) abgefragt werden.

Wenn mehrere Kopien eines Programmes vertrieben werden, können dadurch eindeutige Lizenzmerkmale jeder Kopie angehängt werden, ohne den Quellcode zu ändern.

HINWEIS: Wenn ein Programm mit einer anderen Seriennummer versehen werden soll, muß es neu gelinkt werden.

Für weitere Informationen zu BLINKER EXECUTABLE SERIAL, vergleiche Kapitel 9, für Informationen zu BLISERNUM(), vergleiche Kapitel 10.

Der Blinker Error-Handler

Der Blinker Error-Handler ist eine kleine Routine, die vom Kern bei Auftreten eines Laufzeitfehlers oder bei Ablauf einer Demonstrations-Begrenzung gestartet wird.

Blinker linkt automatisch den bereitgestellten Error-Handler mit in das Programm. Wenn Sie ihn nicht modifiziert haben, gibt es keinen Grund, ihn explizit anzugeben und in Ihr Programm zu linken.

Das Standardverhalten des Error-Handlers äußert sich darin, daß er die Fehlernummer und die dazugehörige Beschreibung anzeigt, wenn ein Fehler auftritt. Nach Anzeige dieser Fehlernachricht wird der Overlay-Manager das Programm beenden und zu DOS zurückkehren.

Der Quelltext des Blinker Error-Handlers liegt in BASIC, C, Pascal und CA-Clipper auf der Original-Diskette vor. Daduch können Sie Modifikationen für Ihre eigenen Zwecke vornehmen.

Modifikationen des Error-Handlers

Der Error-Handler kann an Ihre individuellen Anforderungen angepaßt werden. Vorsicht ist beim Umgang mit den Blinker-Fehlern 1201 (kann Overlay-Datei nicht finden) und 1202 (Fehler beim Lesen der

Overlay-Datei) geboten, da Sie in diesem Fall nicht in der Lage sind, Funktionen im Overlay aufzurufen.

Sollten Sie den Error-Handler modifiziert und rekompiliert haben, muß er in der Skriptdatei mit Hilfe des FILE-Befehls im Rootbereich angegeben werden. Für ein zuverlässiges Arbeiten sollte der Blinker Error-Handler nicht in den Overlay-Bereich gelegt werden. Da bei Auftreten eines Fehlers der Overlay-Manager gehindert wird, Overlays nachzuladen, wird auch der Error-Handler nicht mehr geladen, und kann dann auch keine Fehlermeldungen mehr ausgeben.

Entwicklung unter CA-Clipper

Entwicklung mit CA-Clipper, ein Überblick

Inkrementelles Linken

CA-Clipper 5.x-Debugger

Zusatzbibliotheken von Fremdanbietern

Erzeugen einer DOS Extended CA-Clipper-Anwendung

Erzeugen einer CA-Clipper Windows Anwendung

CA-Clippers DGROUP maximieren mit //INFO

Erzeugen einer CA-Clipper-Anwendung ohne Overlays

Entwicklung mit CA-Clipper, ein Überblick

Die optimale Computer-Umgebung zur Anwendungsentwicklung unterscheidet sich in der Regel von der optimalen Laufzeitumgebung. Daher enthält Blinker mehrere Befehle zur Beschleunigung der Entwicklung, die nur während des Entwickungsprozesses eingesetzt werden sollten.

Es ist sehr wichtig, diese Features vor der Erzeugung der Endversion des ausführbaren Programms wieder zu entfernen, weil das inkrementelle Linken und die eingefügten Debuggerinformationen die Ausführungsgeschwindigkeit dramatisch verringern und die Programmgröße und Speicheranforderungen wesentlich erhöhen können.

Inkrementelles Linken

Einer der bekanntesten Merkmale des Blinker ist die Fähigkeit des inkrementellen Linkens. Das bedeutet, es werden nur die Teile eines Programmes neu gelinkt, die sich seit dem letzten Link-Vorgang geändert haben. Dies ergibt die schnellste Link-Geschwindigkeit und kann die erforderliche Entwicklungszeit einer Applikation drastisch reduzieren.

Die Voreinstellung ist inkrementelles Linken und ist während der Entwicklung transparent. Ein kompletter Link-Vorgang wird nur dann durchgeführt, wenn es notwendig ist, in der Regel nach jedem fünften oder zehnten Link-Vorgang. Dies geschieht ohne Eingreifen des Benutzers.

Die Information, die Blinker benötigt, um inkrementell zu Linken, wird in einer speziellen Datei mit der Erweiterung .BIF gespeichert. Die .BIF-Datei hat normalerweise den gleichen Namen wie die ausführbare Datei, kann aber durch den Entwickler geändert werden. Ein kompletter Link-Vorgang kann zu jeder Zeit durch das Löschen der .EXE- oder .BIF-Datei erzwungen werden.

Wenn inkrementelles Linken angefordert wird, richtet Blinker einen kleinen Puffer in der .EXE-Datei am Ende einer jeden Prozedur oder Funktion ein, um dort Programmänderungen unterzubringen. Sollte dieser zusätzliche Code diesen Puffer überschreiten, wird automatisch ein kompletter Link-Vorgang durchgeführt. Blinker zeigt an, daß ein kompletter Link-Vorgang durchgeführt wurde, sodaß der Entwickler entscheiden kann, ob er diesen Puffer erhöht oder nicht.

Ein Nachteil des inkrementellen Linkens ist, daß sich die Menge des benötigten Speichers erhöht. Das ist das Ergebnis dieser Pufferung und der Tatsache, daß Blinker die CA-Clipper-Symboltabelle nicht komprimiert, wenn inkrementell gelinkt wird.

Um den zusätzlichen Speicherplatz zu entfernen und das Packen der CA-Clipper Symboltabelle zu aktivieren, sollte inkrementelles Linken für den letzten Linklauf vor der Auslieferung des Programms immer ausgeschaltet sein. Dies erreicht man einfach dadurch, daß man den Befehl BLINKER INCREMENTAL OFF an den Anfang des Linkskripts hinzugefügt und neu linkt.

Ein Ausschalten des inkrementellen Linkens bewirkt auch, daß bei Real Mode Programmen jeglicher CA-Clipper Code, sowohl aus .OBJ als auch aus .LIB Dateien und egal, ob innerhalb oder außerhalb von BEGIN/ENDAREA –Bereichen , automatisch überlagert und vom dynamischen Paging-System von CA-Clipper 5.x verarbeitet wird. Das Paging-System bietet eine sehr schnelle und speichereffiziente Überlagerung von CA-Clipper Code mit automatischer Verwendung von erweitertem (expanded) Speicher, falls verfügbar.

Beachten Sie: Während des inkrementellen Linkens werden nur die geänderten .OBJ-Dateien neu gelinkt. Jede Änderung, die an der Skript-Datei zwischen den einzelnen inkrementellen Link-Läufen vorgenommen wurde, hat keine Auswirkung bis zum nächsten kompletten Link-Lauf, außer die Größe der Skript-Datei hat sich geändert; dann erfolgt automatisch ein kompletter Link-Lauf.

CA-Clipper 5.x-Debugger

Um die .OBJ-Dateien für den CA-Clipper-Debugger vorzubereiten, setzen Sie bitte den Compiler-Schalter ‘-b’ ein und vermeiden Sie den Schalter ‘-l’. Der Debugger muß in der Skript-Datei mit dem Befehl FILE <Pfad> CLD.LIB aufgenommen werden. Beachten Sie, daß der FILE-Befehl hierfür verwendet werden muß, und daß Blinker daher die OBJ-Umgebungsvariable nutzt, wenn er nach der CLD.LIB sucht, es sei denn, es wurde ein Pfad bei dem FILE-Befehl angegeben. Wenn die Anwendung läuft, können Sie den Debugger mit ALT-D aktivieren.

```
FILE test
FILE . . .
LIB . . .
FILE C:\CLIPPER\LIB\CLD.LIB   # Beachten Sie den FILE Befehl
```

Achtung: Der selbständig arbeitende (standalone) Debugger CLD.EXE kann nicht im Protected Mode eingesetzt werden und kann daher auch nicht als Debugger für Protected Mode-Programme verwendet werden.

Weitere Informationen hierzu finden Sie im CA-Clipper-Handbuch.

Zusatzbibliotheken von Fremdanbietern

Es gibt zahlreiche Zusatzbibliotheken für CA-Clipper, die den Leistungsumfang der Sprache erweitern. Sehen Sie bitte in den Dokumentationen der einzelnen Produkte nach, um Hinweise zum Linken im Protected Mode zu erhalten.

Wenn man mehrere Zusatzprodukte in einer Anwendung zusammenlinken möchte, muß man vorsichtig sein. Die Reihenfolge dieser Bibliotheken ist wichtig, um eine Rangfolge zu bestimmen. Mögliche Konflikte zwischen diesen Bibliotheken kann man mit dem Befehl BLINKER MESSAGE DUPLICATES erkennen. Damit werden Funktions- und Prozedurnamen sichtbar, die in den Bibliotheken mehrfach vorkommen.

Erzeugen einer DOS Extended CA-Clipper-Anwendung

Beim Linken eines DOS-Extender- oder Dual Mode-Prgramms ist es immer erforderlich, eine zusätzliche Blinker-Bibliothek für den entsprechenden Compiler anzugeben. Diese Bibliothek ersetzt die .OBJ-Dateien in der entsprechenden Compilerbibliothek, die nicht kompatibel zum Protected Mode sind, durch neue .OBJ-Dateien.

All diese Blinker-Bibliotheken haben einen Namen in der Form BLXcccnn. Dabei steht ccc für die Firma und den Compiler, z.B. CLP für CA-Clipper, und nn verweist auf die Versionsnummer des Compilers. Die jeweilige Blinker Protected Mode Compiler-Bibliothek muß über den Befehl SEARCH immer genau vor der Standardbibliothek des Compilers oder der Skriptdatei mit dieser Standardbibliothek angegeben werden. Wenn keine Compiler-Bibliothek explizit angegben wurde, dann fügen Sie den SEARCH-Befehl für die Blinker Bibliothek an das Ende der Bibliotheksliste in den Linkskript ein.

Beispiel für CA-Clipper 5.3:

```
SEARCH BLXCLP53  # Blinker-Bibliothek
LIB CLIPPER         # CA-Clipper-Bibliothek
```

Werden Drittanbieter-Bibliotheken verwendet, dann achten Sie bitte darauf, daß diese den Protected Mode unterstützen, bevor Sie versuchen, diese in einem DOS Extended Programm einzusetzen. Blinkinc kann nicht dafür garantieren, daß eine bestimmte Bibliothek overlayfähig ist oder im Protected Mode eingesetzt werden kann oder gar beides. Daher erkundigen Sie sich bitte beim Anbieter der jeweiligen Bibliothek nach der Kompatibilität zum Protected Mode.

Erzeugen einer CA-Clipper Windows Anwendung

Blinker kann zusammen mit den Produkten Clip-4-Win, FiveWin und CLProfiW benutzt werden, um CA-Clipper 5.x Windows Programme zu erzeugen. Bitte schauen Sie in die Dokumentation dieser Produkte für weitere Hinweise über die Verwendung des Blinkers.

Das OBJ Unterverzeichnis des Blinker Installationsverzeichnisses enthält die Datei MPAR.OBJ, welche zu allen CA-Clipper-für-Windows Programmen gelinkt werden sollte. Diese Datei fixt einen Bug in CLIPPER.LIB Version 5.3b und früher, der Windowsprogramme daran hinderte, zu laden, wenn eine bestimmte interne Tabelle eine 32 KB Grenze überschritten hatte. Dies konnte bei einem CA-Clipper Windowsprogramm beliebiger Größe passieren, und in dem Falle würde Windows meistens die hilfreiche Meldung 'Ein Gerät, das mit dem System verbunden ist, antwortet nicht' ausgeben und das Programm beenden.

CA-Clippers DGROUP maximieren mit //INFO

CA-Clipper verwendet zur Laufzeit eines Programms möglichst viel von der Standard Datengruppe (DGROUP), um dort temporäre Werte, z.B. für Locals, Statics und Funktionsparameter, zu speichern. Der berüchtigte ‘667 Eval stack fault’ Fehler erscheint, wenn es dort nicht ausrechen Speicher für den Bedarf des Programmes gibt, was immer wahrscheinlicher wird, je größer das Programm ist.

Wenn man beim Aufruf eines CA-Clipperprogramms den String //INFO an die Befehlszeile anhängt, dann zeigt CA-Clipper beim Start und Beim Beenden des Programms summarisch die Verwendung des DGROUP an. Für die Bedeutung der einzelnen angezeigten Werte sehen Sie bitte in der CA-Clipper Dokumentation nach.

Die wichtigste Zahl, was den Eval Stack angeht, ist die ‘DS avail’ Zahl, die ungefähr angibt, wieviel Speicher im Standard Datensegment für den Eval Stack zur Verfügung steht. Durch Hinzufügen und Entfernen von Programmteilen kann man mit dieser Zahl bestimmen, welche Fremdbibliothek (wenn überhaupt) oder welcher Abschnitt im CA-Clipper Code den Speicher aufbraucht.

Beachten Sie, daß eine Vergrößerung des System Stacks (mit STACK nnn oder BLINKER PROCEDURE DEPTH nnn) die Zahl für ‘DS avail’ erniedrigt und damit auch die Größe des Eval Stack, weil mehr DGROUP Speicher für den System Stack verbraucht wurde.

größerer Eval Stack in DOS Extended Programmen

CA-Clippers DGROUP maximieren mit //INFO

größerer Eval Stack in DOS Extended Programmen

Beim Linken von DOS Extended Programmen fügt Blinker 5.0 und höher automatisch einen Code hinzu, um den verfügbaren Speicher für den Eval Stack dadurch stark zu erhöhen, daß ein größerer Teil nach außerhalb der DGROUP verlagert wird. Damit sollte der berüchtigte 667 Fehler nicht mehr vorkommen, da alle statischen Variablen komplett aus dem Eval Stack entfernt werden und somit wesentlich mehr Platz übrig bleibt für die anderen Informationen, die CA-Clipper dort speichern muß.

Der //INFO Parameter wird weiterhin den selben 'DS avail' Wert anzeigen, wie bei früheren Versionen. Aber es gibt noch zwei zusätzliche Zeilen, die den verfügbaren Speicher für die beiden Haupbestandteile des Eva Stacks anzeigen, nämlich den Speicher für LOCALs und PRIVATEs, sowie den Speicher für STATICs. Z.B.:

```
Clipper (R) 5.3b Intl. (Rev. 338) US - ASCII Collation
DS=0297:0000  DS avail=38KB  OS avail=1023KB  EMM avail=0KB

LOCALs and PRIVATEs space = 37KB    Used = 1KB    Remaining = 36KB
STATICs space             = 63KB    Used = 0KB    Remaining = 63KB

*** Program Output ***

LOCALs and PRIVATEs space = 37KB    Used = 10KB    Remaining = 27KB
STATICs space             = 63KB    Used = 15KB    Remaining = 48KB

(Fixed heap=25KB/1)
```

Wenn das Programm endet, haben die LOCALs und PRIVATEs noch 27 KB Speicher von insgesamt 37 KB zur Verfügung, während die STATICs noch 48 KB von ihren eigenen 63 KB übrig haben.

Dieser Eval Stack Zusatzcode ist staqndardmäßig aktiv, kann aber einfach abgeschaltet werden, um die relativen Verbesserungen mit und ohne zu vergleichen oder zum Debuggen. Um diesen Zusatzcode auszuschalten, starten Sie das Programm mit //NOEVAL als Kommandozeilenparameter (normalerweise zusätzlich zum //INFO Parameter). Z.B.:

```
Clipper (R) 5.3b Intl. (Rev. 338) US - ASCII Collation
DS=0297:0000  DS avail=38KB  OS avail=1023KB  EMM avail=0KB

LOCALs and PRIVATEs space = 37KB    Used = 1KB    Remaining = 36KB
STATICs space * SHARED *  = 36KB    Used = 0KB    Remaining = 36KB

*** Program Output ***

LOCALs and PRIVATEs space = 37KB    Used = 10KB    Remaining = 12KB
STATICs space * SHARED *  = 36KB    Used = 15KB    Remaining = 12KB

(Fixed heap=25KB/1)
```

Wenn das Programm jetzt endet, teilen sich alle LOCALs, PRIVATEs und STATICs die gleichen 37 KB an Speicher, wie 'DS avail' angibt, und es bleiben nur 12 KB übrig für alle Variablentypen. Das ist so bei allen Linkern und allen Versionen von Blinker vor 5.0 .

CA-Clippers DGROUP maximieren mit //INFO

MSGSYSnn.ZIP Ersatz-Fehlermeldungen

Im CLP Unterverzeichnis des Blinker Installationsverzeichnisses gibt es eine Datei MSGSYSnn.ZIP. Diese enthält die erforderlichen Dateien, um nationale Versionen der verschiedenen Clipper Fehlermeldungen zu erzeugen und sie gleichzeitig aus der DGROU P zu entfernen, um damit weitere 1.5 KB an wertvollem 'DS avail' Speicher einzusparen. Diese Datei wurde freundlicherweise von Matteo Baccan von der Italian Software Agency zur Verfügung gestellt.

Beachten Sie bitte, daß Blinkinc diese Dateien nur als Dienst an CA-Clipper Anwender verteilt und sie so liefert, wie sie sind, und daher auch keine technische Unterstützung und Hilfe für den Inhalt dieser Dateien geben kann und wird. Bitte richten Sie alle Fragen und Kommentare an direkt an Matteo Baccan über baccan@isanet.it.

Erzeugen einer CA-Clipper-Anwendung ohne Overlays

Manchmal ist es erforderlich, eine CA-Clipper-Anwendung ohne Overlays zu erzeugen. Das ist zum Beispiel für ein Installationsprogramm der Fall, für das mehrere Disketten benötigt werden und das daher nicht zu der .EXE-Datei zurückkehren kann, was aber für die Overlays erforderlich wäre. In solch einem Fall müssen folgende zwei Befehle eingesetzt werden:

```
BLINKER INCREMENTAL OFF    // einem BLI INC OFF muß
BLINKER CLIPPER PAGE OFF      // ein BLI CLI PAG OFF folgen,
                           // damit es fehlerfrei funktioniert
```

Weiterhin müssen aus der Link-Skriptdatei alle BEGIN/ENDAREA-Befehle entfernt werden.

Der Befehl BLINKER INCREMENTAL OFF komprimiert die Symboltabelle und löscht die Zwischenräume zwischen den Prozeduren wie gewöhnlich. Danach aber muß sofort der Befehl BLINKER CLIPPER PAGE OFF folgen, um das automatische Einbinden des CA-Clipper-Code mit Hilfe des dynamischen Paging Systems von CA-Clipper in Overlays durch den Blinker zu verhindern.

Link-Skriptbefehle

Befehle - Überblick

Kurz-Übersicht der Blinker-Befehle

Ausführliche Befehlsreferenz

Befehle - Überblick

Dieser Abschnitt dient als Überssicht für alle Blinker Skriptbefehle, aber man sollte diese Befehle schon gründlich verstehen, bevor man versucht, ein Programm zu erzeugen oder ein bestehendes Programm für Blinker zu konvertieren.

Skriptdatei(en) werden in der Befehlszeile eingegeben und haben die vorgegebene Dateinamenerweiterung .LNK. Skriptdateien können ineinander verschachtelt werden und es können mehrere Skriptdateien in der Kommandozeile verarbeitet werden, z.B.:

```
BLINKER @STANDARD @ANDERE LIB Mylib VERBOSE
```

Diese Befehlszeile gibt Blinker die Anweisung, zuerst die Skriptdatei STANDARD.LNK und daran anschließend die Skriptdatei ANDERE.LNK einzulesen und zu verarbeiten. Weiterhin erhält Blinker die Anweisung, die MYLIB.LIB zu verwenden und ein VERBOSE-Listing der verarbeiteten .OBJ-Module auszugeben.

In der Skriptdatei können entweder der vollständige Befehlsname oder eine eindeutige Abkürzung verwendet werden, obwohl aus Gründen der Übersichtlichkeit mindestens drei Buchstaben des Befehlsnamens verwendet werden sollten. Der Befehl

```
BLINKER OVERLAY OPSIZE
```

kann beispielsweise abgekürzt werden als :

```
BLI OVE OPS
```

Kurz-Übersicht der Blinker-Befehle

Die Buchstaben R, P und W in den ersten Spalten zeigen jeweils an, ob der Befehl auf ein Real Mode Programm, Protected Mode Programm oder Windows Programm anwendbar ist.

Allgemeine Befehle

R P W # (Kommentar)
Ein Kommentar in der Skriptdatei
R P W // (Kommentar)
Ein Kommentar in der Skriptdatei
R P W @<Skriptname>
Der Name der Skriptdatei, die Blinker verarbeiten soll
R P W BATCH
Batch-Modus angeben
R BEGINAREA
Beginn des dynamischen Overlay-Bereichs
R BLINKER CACHE EMS <nuCache>
Größe des EMS-Bereichs, der für die Overlay-Pufferung verwandt werden soll
R BLINKER CACHE XMS <nuCache>
Größe des XMS-Bereichs, der für die Overlay-Pufferung verwandt werden soll
R P BLINKER DEBUG HANDLE <nuHandle>
Filehandle für die Debugging-Ausgabe
Standard: 1
R P BLINKER DEMONSTRATION DATE <jjjj/mm/tt>
Verfallsdatum einer Demo-Version
R P BLINKER DEMONSTRATION MINUTES <nuLimit>
Maximale Laufzeit einer Demo-Version in Minuten
R P BLINKER ENVIRONMENT NAME <cName>
Verändert den Namen der Blinker-Umgebungsvariable
Standard: BLINKER
P W BLINKER EXECUTABLE ALIGNMENT <nuFileAlign> [,nuSecAlign]
Ausrichtungsgröße für eine neue .EXE-Datei
R P W BLINKER EXECUTABLE COMPRESS [<nuFaktor>]
Komprimierungsgrad für eine .EXE-Datei
R BLINKER EXECUTABLE DUAL [<nuGröße>]
Erzeugen einer im Dual Mode ausführbaren Datei
P BLINKER EXECUTABLE EXTENDED [<nuGröße>]
Erzeugen eines DOS-Extender-Programms
R P BLINKER EXECUTABLE IPX <nuECBst> [,<nuBuffers>]
Anzahl der ECBs und Puffer für IPX/SPX Unterstützung.
Standard: 64,32
R BLINKER EXECUTABLE NOBLINK
Kein Blinker-Code in der .EXE
R P BLINKER EXECUTABLE NODELETE
Eine .EXE-Datei wird ohne Rücksicht auf unaufgelöste externe Referenzen erstellt
R BLINKER EXECUTABLE REAL
Erzeugen einer im Real Mode ausführbaren Datei
R P BLINKER EXECUTABLE SERIAL <cSeriennummer>
Angabe einer verschlüsselt in der .EXE-Datei zu speichernden Seriennummer
P BLINKER EXTMEM BLOCK <nuGröße>
Blockgröße für Erweiterungsspeicher setzen
P BLINKER EXTMEM LIMIT <nuLimit>
Angabe des maximal zu verwendenden Erweiterungsspeichers
P BLINKER HOST DPMI ON|OFF
DPMI-Host-Standard für den Protected Mode einschalten
Standard: ON
P BLINKER HOST MESSAGE ON|OFF

Statusmeldung beim Programmstart einschalten
Standard: OFF

P BLINKER HOST QDPMI ON|OFF
QDPMI-Host-Standard für den Protected Mode einschalten
Standard: ON

P BLINKER HOST VCPI ON|OFF
VCPI-Host-Standard für den Protected Mode einschalten
Standard: ON

P BLINKER HOST XMS ON|OFF
XMS-Host-Standard für den Protected Mode einschalten
Standard: ON

R P BLINKER LINK EMS ON|OFF
Einstellen, ob EMS-Speicher zur Linkzeit verwendet werden soll oder nicht
Standard: ON

R P BLINKER LINK PAGEFRAME ON|OFF
Einstellen, ob die EMS-Pageframe zur Linkzeit verwendet werden soll oder nicht

R P W BLINKER LINK VERSION <n.n>
Spezifizierung der minimalen Blinker-Version;
Standard: 0.0

R P BLINKER LINK XMS ON|OFF
Einstellen, ob XMS-Speicher zur Linkzeit verwendet werden soll oder nicht
Standard: ON

P BLINKER LOWMEM RESERVE <nuMemSize>
konventionellen Speicher für Programm reservieren

R BLINKER MEMORY CLEAR <nuWert>
Überschreibt den Speicher mit einem Wert, bevor das Programm geladen wird

R P W BLINKER MESSAGE DUPLICATES
Warnungen erlauben, falls doppelte Module in Bibliotheken gefunden werden

R P BLINKER MESSAGE NOBLINK
Das Blinken der Augen während des Linklaufs abschalten

R P W BLINKER MESSAGE NOWARNING
Die Anzeige von Warnungen während des Linklaufs abschalten

R P BLINKER MESSAGE WINK [LEFT]
Einschalten des Blinkens mit nur einem Auge

R BLINKER OVERLAY OPSIZE <nuGröße>
Die Arbeitsgröße des Overlay-Bereichs festlegen
Standard: 40 KB

R BLINKER OVERLAY PAGEFRAME ON|OFF
Zugriff auf die EMS-Pageframe durch den Overlay-Bereich erlauben
Standard: OFF

R BLINKER OVERLAY THRESHOLD <nuGröße>
Die Größe des kleinsten Overlay-Segments festlegen Standard: 32 Bytes

R BLINKER OVERLAY UMB ON|OFF
Benutzung von XMS UMBs für den Overlay-Bereich
Standard: ON

P W BLINKER SEGMENT THRESHOLD <nuGröße>
Größe des größten zusammengesetzten Segments der neuen .EXE-Datei bestimmen
Standard: 65500 Bytes

R P W CODECLASS
Zusätzliche Segmentklassen angeben, die wie CODE behandelt werden sollen

R P W CODEVIEW

Einschalten der CodeView Debugger-Unterstützung; Standard: OFF

R P W CVOFF
Die Verarbeitung von CodeView-Informationen lokal abschalten

R P W CVON
Die Verarbeitung von CodeView-Informationen lokal einschalten

R P W CVVERSION <nuLevel>
Die Versionsnummer der CodeView-Informationen einstellen
Standard: 5

P W DEFBEGIN
Moduldefinitionsbereich in einer Skriptdatei beginnen

P W DEFEND
Modukdefinitionsbereich in einem Linkskript beenden

P W DEFFILE <Dateiname>
Angabe einer externen Moduldefinitionsdatei

R P W DEFINE <cSymbol> [, <cSymbol> ...]
Zur Linkzeit ausgeschlossene Symbole definieren

R P W DEFLIB
Verwendung der vorgegebenen Bibliothek-Sucheinträge
Standard: aktiviert

R P W DOSSEG
Verwendung der Segmentanordnung von Microsoft; Standard: deaktiviert

R P W ECHO [ON|OFF] [<cZeichenkette>]
Skript-Kommandos auf dem Bildschirm ausgeben
Standard: OFF

R ENDAREA
Einen Overlay-Bereich beenden

R P W EXTDICTIONARY
Verwendung des Bibliotheksverzeichnisses aktivieren
Standard: aktiviert

R EXTRAMEM <nuGröße>
Beschränkung der Speicherbelegung

R P W FILE [L:][Pfad]<Dateiname> [, [L:][Pfad]<Dateiname> ...]
Angabe einer oder mehrerer Objekt-Dateien

R P W LIBRARY [L:][Pfad]<Bibliotheksname>
[, [L:][Pfad]<Bibliotheksname> ...]
Angabe einer oder mehrerer Eingabe-Bibliotheken

R P W MAP [=<Dateiname>] [A] [,S] [,X]
Erstellen einer Map-Datei zu der erzeugten .EXE;
Standard: A (falls MAP angegeben wurde)

R MEMORY <nuLimit>
Beschränkung der Speicherbelegung

R P W MIXCASE [EXPORT | IMPORT]
Unterscheidung von Groß-/Kleinschreibung für Symbole während des Linklaufs
Standard: Deaktiviert

R P W MODULE <Modul> [,<Modul> [,...]] [FROM <Bibliotheksname>]
Die Position bestimmter Bibliotheksmodule festlegen

R MURPHY
Eine "worst case"-Overlaysituation erzeugen;
Standard: Deaktiviert

R P W NOBELL
Den Ton nach dem Ende eines Linklaufs unterdrücken

R P W NODEFLIB
Nicht nach Standard-Bibliotheken suchen
Standard: Deaktiviert

R P W NOEXTDICTIONARY
Unterdrücken der Bearbeitung von erweiterten Bibliothek-Verzeichnissen
Standard: Deaktiviert

P W NOPACKCODE
Deaktivieren des Packen von Codesegmenten
Standard: Aktiviert

P W NOPACKDATA
Deaktivieren des Packens von Datensegmenten
Standard: Deaktiviert, außer für CA-Clipper 5.x mit aktivierter Symboltabellenkompression

R P W NOTABLEOFCONTENTS
Das Durchsuchen (SEARCH) aller Bibliotheken im Skript erzwingen.
Standard: Deaktiviert

R P W NWIDTH <nuWeite>
Die Breite von Symbolnamen in der Map-Datei festlegen.
Standard: 23

R P W OUTPUT [L:][Pfad]<Dateiname>
Angabe des Namens für das ausführbare Programm Standard: <erste Objektdatei>.EXE

R OVERLAY <cKlassenname> [,<cKlassenname> ...]
Angabe, welche Segment-Klassen ins Overlay gelegt werden sollen
Standard: CODE

P W PACKCODE [<nuMaxSegGröße>]
Codesegmente beim Linken packen
Standard: 65500

P W PACKDATA [<nuMaxSegGröße>]
Datensegmente beim Linken packen
Standard: 65535

W RC <cResdateiname>
Anfügen einer Resourcedatei an die EXE oder .DLL

R P W READONLY
Attribut des ausführbaren Programms auf ‘Nur lesen’ setzen.
Standard: Deaktiviert

R P W SEARCH [L:][Pfad]<Bibliotheksname>
[, [L:][Pfad]<Bibliotheksname> ...]
Bibliotheken, die komplett nach Symbolen durchsucht werden sollen.

R SECTION INTO <Overlayname>
[FILE [L:][Pfad]<Datei> [, [L:][Pfad]<Datei>...]]
Namen der externen Overlays und der Dateien, die darin plaziert werden sollen

R P STACK <nuGröße>
Größe des Programm-Stacks (dezimale Anzahl der Bytes)

R P W UPPERCASE
Alle Symbole während des Linkens in Großschrift verwandeln
Standard: aktiviert

R P W VERBOSE [<nuLevel>]
Zeigt Statusinformationen während des Linkens an Standard: Deaktiviert

R P W WORKFILE [L:][Pfad]<Dateiname>
Gibt Verzeichnis und Namen der temporären Arbeitsdatei an

CA-Clipper-spezifische Befehle

Bitte beachten Sie: Sollten Sie irgendwelche Änderungen an der Skriptdatei vorgenommen haben, so muß ein kompletter Linklauf durchgeführt werden, damit die Modifikationen auch wirksam werden. Sobald inkrementelles Linken eingeschaltet ist, wird ein kompletter Linklauf automatisch durchgeführt, sofern sich die Größe der Skriptdatei geändert hat. Sollte die Größe nach einer Veränderung nicht differieren, so können Sie das inkrementelle Linken ausschalten oder die .BIF- bzw. die .EXE-Datei löschen, um einen kompletten Linklauf zu erzwingen.

Beachten Sie, daß CA-Clipper alleine keine Windows Programme erzeugen kann, sodaß auch Blinker nur dann CA-Clipper Windows Programmme erzeugen kann, wenn Zusatzprodukte wie Clip4Win oder FiveWin verwendet werden.

R	P		BLINKER CLIPPER PAGE OFF Deaktiviert das Dynamische Paging-System von CA-Clipper oder erzeugt eine .EXE-Datei ohne Overlays Standard: ON
R	P	W	BLINKER ENVIRONMENT CLIPPER <cName> Namen der CA-Clipper-Umgebungsvariablen verändern
R	P	W	BLINKER ENVIRONMENT OVERRIDE Gestattet, daß durch den SET CLIPPER Befehl die Einstellung BLINKER EXECUTABLE CLIPPER zur Laufzeit überschrieben wird.
R	P	W	BLINKER EXECUTABLE CLIPPER <cUmgebungsstring> Standard-CA-Clipper-Umgebungseinstellungen angeben
R	P	W	BLINKER INCREMENTAL FILE [L:][Pfad]<Dateiname> Der Dateiname der Inkrementaldatei Standard: <.EXEname>.BIF
R	P	W	BLINKER INCREMENTAL ON|OFF Ein-/Ausschalten des inkrementellen Linkens Standard: ON
R	P	W	BLINKER INCREMENTAL PAD <nuGröße> Angabe der Größe für Einfügeoperationen des inkrementellen Linkens Standard: 128 Bytes
R			BLINKER MEMORY PACK <nuFrequenz> Angabe der Speicherpack-Frequenz - gilt nur für Sommer ’87
R			BLINKER OVERLAY DYNAMIC Angabe, ob die dynamische Overlay-Belegungsmethode verwendet werden soll - gilt nur für Sommer ’87 Standard: Aktiviert für CA-Clipper Sommer ‘87
R			BLINKER OVERLAY FIXED Angabe, ob die statische Overlay-Belegungsmethode verwendet werden soll Standard: Deaktiviert für CA-Clipper Sommer ‘87
R	P		BLINKER PROCEDURE DEPTH <nuTiefe> Angabe der maximalen Prozedurverschachtelungstiefe Standard: 50

Ausführliche Befehlsreferenz

Dieser Abschnitt beschreibt alle Linkskriptbefehle von Blinker im Detail.

(Kommentar)

// (Kommentar)

@ (verschachtelte Skriptdateien)

BATCH

BEGINAREA

BLINKER CACHE EMS

BLINKER CACHE XMS

BLINKER DEBUG HANDLE

BLINKER DEMONSTRATION DATE

BLINKER DEMONSTRATION MINUTES

BLINKER ENVIRONMENT NAME

BLINKER EXECUTABLE ALIGNMENT

BLINKER EXECUTABLE CLIPPER

BLINKER EXECUTABLE COMPRESS

BLINKER EXECUTABLE DUAL

BLINKER EXECUTABLE EXTENDED

BLINKER EXECUTABLE IPX

BLINKER EXECUTABLE NOBLINK

BLINKER EXECUTABLE NODELETE

BLINKER EXECUTABLE REAL

BLINKER EXECUTABLE SERIAL

BLINKER EXTMEM BLOCK

BLINKER EXTMEM LIMIT

BLINKER HOST DPMI

BLINKER HOST MESSAGE

BLINKER HOST PASSUP

BLINKER HOST QDPMI

BLINKER HOST VCPI

BLINKER HOST XMS

BLINKER LINK EMS

BLINKER LINK PAGEFRAME

BLINKER LINK VERSION

BLINKER LINK XMS

BLINKER MEMORY CLEAR

BLINKER MESSAGE DUPLICATES

BLINKER MESSAGE NOBLINK

BLINKER MESSAGE NOWARNING

BLINKER MESSAGE WINK

BLINKER OVERLAY OPSIZE

BLINKER OVERLAY PAGEFRAME

BLINKER OVERLAY THRESHOLD

BLINKER OVERLAY UMB

BLINKER SEGMENT THRESHOLD

CODECLASS

CODEVIEW

CVOFF

CVON

CVVERSION

DEFBEGIN

DEFEND

DEFFILE

DEFINE

DEFLIB

DOSSEG

ECHO

ENDAREA

EXTDICTIONARY

EXTRAMEM

FILE

LIBRARY (Linkskriptbefehl)

MAP

MEMORY

MIXCASE

MODULE

MURPHY

NOBELL

NODEFLIB

NOEXTDICTIONARY

NOPACKCODE

NOPACKDATA

NOTABLEOFCONTENTS

NWIDTH

OUTPUT

OVERLAY

PACKCODE

PACKDATA

RC (Linkskriptbefehl)

READONLY

SEARCH

SECTION INTO

STACK

UPPERCASE

VERBOSE

WORKFILE

(Kommentar)

Zweck: Ein Skriptdatei-Kommentar

Syntax:
```
# ...
```

Beispiel:
```
# Linken oder nicht linken, das ist die Frage.
```

Bemerkung: Der Reste aller Zeilen, die in der Skriptdatei ein #-Zeichen enthalten, werden als Kommentar behandelt und somit von Blinker ignoriert.

Siehe auch: // (Kommentar)

// (Kommentar)

Zweck: Ein Skriptdatei-Kommentar

Syntax:
```
// ...
```

Beispiel:
```
// Linken oder nicht linken, das ist die Frage.
```

Bemerkung: Alle Zeilen, die in der Skriptdatei mit einem //-Zeichen beginnen, werden als Kommentarzeilen behandelt und somit von Blinker ignoriert.

Siehe auch: # (Kommentar)

@ (verschachtelte Skriptdateien)

Zweck: Angabe des Namens einer Skript-Datei

Syntax:
```
@<Skript-Dateiname>
```

Beispiel:
```
# Skriptdatei Befehle
...
# Führe das Skript 'Meinskript' aus
@Meinskript
# fahre mit weiteren Skriptbefehlen fort
...
```

Bemerkung: Blinker unterstützt verschachtelte Skript-Dateien beliebiger Größe bis zu einer maximalen Tiefe von fünf Stufen. Die Verarbeitung einer geschachtelten Skript-Datei ähnelt einem Funktionsaufruf: trifft Blinker auf einen Befehl für ein verschachteltes Skript, so wird das aktuell bearbeitete Skript in seiner Verarbeitung angehalten, das neue Skript wird bearbeitet und danach kehrt Blinker wieder zu dem ursprünglichen Skript zurück und fährt mit diesem fort.

Wenn Blinker nach einer Skript-Datei sucht, so tut er dies zunächst in dem aktuellen Verzeichnis. Anschließend wird der Pfad durchsucht, der durch den 'SET LIB'-Befehl angegeben wurde. Dies ermöglicht es, häufig benötigte Skript-Dateien, die selten

modifiziert werden (wie Skripts, die das Overlaying einer Standard-Bibliothek beschreiben), aus praktischen Gründen in einem Verzeichnis unterzubringen.

BATCH

Zweck: Blinker mitteilen, daß er sich in einer Stapelausführung befindet

Syntax: `BATCH`

Standard: ausgeschaltet

Bemerkung: Der BATCH-Befehl teilt Blinker mit, daß er sich in einem Stapel-Modus befindet. Dies bedeutet, daß bei einem auftretenden Fehler der Linker die Arbeit mit einem Fehler abbricht, anstatt in den interaktiven Modus zu schalten. Blinker ignoriert den BATCH-Befehl, da dies der Standardeinstellung des Blinkers entspricht.

BEGINAREA

Zweck: Bestimmt den Anfang eines dynamischen Overlay-Bereichs

Syntax: `BEGINAREA`

Bemerkung: Der Befehl BEGINAREA wird verwendet, um dem Linker den Beginn eines Overlay-Bereiches in der Ausgabedatei anzuzeigen. Jedes BEGINAREA muß ein korrespondierendes ENDAREA besitzen, welches das Ende des Overlay-Bereiches bestimmt.

Bei statischen Overlays können BEGINAREA's, d.h. Overlay-Bereiche, ineinander verschachtelt werden, soweit immer das zugehörige ENDAREA mit angegeben worden ist. Dies ermöglicht dem Entwickler, eine Overlay-Struktur für einen Bereich zu erzeugen, in dem eine minimale Menge an Speicherplatz für eine maximale Anzahl von Overlay-Routinen verwendet wird.

Bei den dynamischen Overlays von Blinker sind, im Gegensatz zu statischen Overlays, die Abhängigkeiten der einzelnen Overlay-Prozeduren oder Dateien untereinander unwichtig. Auch die im Zusammenhang mit statischen Overlays vorkommenden mehrfach verschachtelten Overlay-Bereiche sind nicht mehr nötig, obwohl mehrere Overlay-Bereiche zur Ordnung der einbezogenen Bibliotheken nützlich sein können. Die Befehle BEGINAREA und ENDAREA werden schlicht als Markierungen für Anfang und Ende des Codes betrachtet, der als Overlay angelegt werden soll.

Bei den meisten Compilern werden die zwischen BEGINAREA und ENDAREA aufgelisteten .OBJ-Dateien zur Laufzeit auf Segmentbasis verwaltet, d. h. das gesamte CODE-Segment für eine .OBJ wird in den Speicher geladen, sobald eine beliebige Funktion oder Prozedur innerhalb dieser .OBJ-Datei aufgerufen wird.

Wenn Sie weitere Details darüber erfahren möchten, wie Funktionen und Prozeduren in separaten Segmenten gehalten werden können, um die Größe einer Overlay-Einheit zu reduzieren, konsultieren Sie bitte das Compiler-Handbuch der jeweiligen Sprache.

Hinweis für CA-Clipper:

Alle zwischen BEGINAREA und ENDAREA aufgelisteten CA-Clipper Sommer '87-.OBJ-Dateien werden während der Laufzeit auf einer Prozedur- bzw. Funktionsbasis bearbeitet. Hierbei ist es gleich, wieviele Funktionen oder Prozeduren in einer .OBJ vorhanden sind. Darum sollten extrem große Prozeduren oder Funktionen vermieden werden. Das dynamische Paging-System von CA-Clipper 5.x zerlegt die Prozeduren automatisch in Seiten zu je 1 KB Größe, was sich vorteilhaft auf die Ausführungsgeschwindigkeit auswirkt.

Siehe auch: ENDAREA
SECTION INTO

BLINKER CACHE EMS

Zweck: Bestimmt die Menge an EMS-Speicher, die für Overlay-Caching benutzt werden soll.

Syntax: `BLINKER CACHE EMS <nuMax[%]>, <nuMinLeave[%]>`

Standard: <nuMax> ist 100KB
<nuMinLeave> ist 0 KB

Beispiel:
```
# nutze maximal 50% des verfügbaren EMS-Speichers, aber
# lasse mindestens 1024 KB frei
BLINKER CACHE EMS 50%,1024
```

oder

```
# Nutze maximal 512 KB des verfügbaren EMS-
#  Speichers, aber lasse mindestens 25% frei
BLINKER CACHE EMS 512, 25%
```

Bemerkung: Dieser Befehl legt die Menge des EMS-Speichers (ab EMS 4.0) fest, die Blinker für die Overlay-Pufferung bereitstellen soll.

Es werden zwei numerische Parameter angegeben, die die Größe des zu reservierenden EMS-Speichers festlegen, bzw. die Größe des EMS-Speichers, der zur freien Verfügung stehen soll. Beide Parameter können sowohl in Prozent als auch in Kilobyte angegeben werden. Der zweite Parameter überschreibt den ersten, d.h., wenn der erste Parameter den Wert 256 KB angibt und der zweite Parameter den Wert 50% (also 50% des EMS freibleiben sollen), dann wird die Hälfte des EMS nicht genutzt. Der Cache paßt sich an den verbleibenden Speicher an.

Blinker wird nie einen Cache belegen, der größer als die Gesamtgröße aller im Programm enthaltenen Overlays ist.

Informationen über die Ermittlung der optimalen Cache-Größe finden Sie im Anhang A "MAP-Dateiausgabe".

Siehe auch: BLINKER CACHE XMS
BLINKER ENVIRONMENT NAME

BLINKER CACHE XMS

Zweck: Bestimmt die Menge an XMS-Speicher, die für Overlay-Caching benutzt werden soll.

Syntax: `BLINKER CACHE XMS <nuMax[%]>, <nuMinLeave[%]>`

Standard: <nuMax> ist 100KB
<nuMinLeave> ist 0 KB

Beispiel:
```
# Nutze 50% des verfügbaren XMS-Speichers, aber
#  lasse mindestens 1024 KB frei
BLINKER CACHE XMS 50%, 1024
```

oder

```
# Nutze maximal 512 KB des verfügbaren XMS-
#  Speichers, aber lasse mindestens 25% frei
BLINKER CACHE XMS 512, 25%
```

Bemerkung: Dieser Befehl legt die Menge des XMS (ab Version 2.0)-Speichers fest, die Blinker für die Overlay-Pufferung bereitstellen soll.

Es können zwei numerische Parameter angegeben werden, die die Größe des zu reservierenden XMS-Speichers festlegen, bzw. die Größe des XMS-Speichers, die zur freien Verfügung stehen soll. Beide Parameter können sowohl als Prozent-Angabe, wie auch als Kilobyte-Angabe angegeben werden. Der zweite Parameter überschreibt den ersten, d.h., wenn der erste Parameter den Wert 256 KB angibt und der zweite Parameter den Wert 50% (also 50% des XMS freibleiben sollen), dann wird die Hälfte des XMS nicht genutzt. Der Cache paßt sich an den verbleibenden Speicher an.

Blinker wird nie einen Cache belegen, der größer als die Gesamtgröße aller im Programm enthaltenen Overlays ist.

Informationen über die Ermittlung der optimalen Cache-Größe finden Sie im Anhang A "MAP-Dateiausgabe".

Siehe auch: BLINKER CACHE EMS
BLINKER ENVIRONMENT NAME

BLINKER CLIPPER PAGE

Zweck: Aus-/Einschalten des automatischen Codepagings von CA-Clipper 5.x

Syntax: `BLINKER CLIPPER PAGE ON | OFF`

Standard: ON (wenn BLINKER INCREMENTAL OFF)
OFF (wenn BLINKER INCREMENTAL ON)

Bemerkung: Dieser Befehl dient zum Ausschalten des dynamischen Seitenaustauschs für CA-Clipper-Code bei CA-Clipper 5.x. Der einzige Grund, wann dies gewünscht sein kann, ist, wenn das CA-Clipper 5.x Programm keine Overlays haben darf, z.B. um ein Installationsprogramm zu schreiben. Das Paging-System ist durch Blinker automatisch dann ausgeschaltet, wenn inkrementelles Linken eingeschaltet ist und umgekehrt.

Wird mit BLINKER INCREMENTAL OFF gearbeitet, so ist das Paging-System automatisch eingeschaltet, wenn nicht dieser Befehl benutzt wird.

Es wird empfohlen, das CA-Clipper 5.x Paging-System in allen CA-Clipper 5.x-Applikationen zu verwenden, da es meistens wesentlich schneller arbeitet als gewöhnliches, dynamisches Overlaying.

Hinweis: Durch die Verwendung des Befehls BLINKER CLIPPER PAGE OFF wird die Symboltabelle komprimiert und sämtlicher CA-Clipper-Code wird automatisch in Overlays abgelegt (wie bei dem Befehl BLINKER INCREMENTAL OFF). Diese Overlays werden jedoch vom Blinker und nicht durch das Dynamische Paging-System von CA-Clipper verwaltet.

Siehe auch: BLINKER INCREMENTAL OFF

BLINKER DEBUG HANDLE

Zweck: Gibt das File-Handle für die Ausgabe der Debugging-Routinen an

Syntax: `BLINKER DEBUG HANDLE <nuHandle>`

Standard: 1 (Bildschirmausgabe)

Beispiel:
```
# setze Standard-Ausgabe auf Drucker
BLINKER DEBUG HANDLE 4
```

Bemerkung: Dieser Blinker-Befehl setzt die Voreinstellung für das Datei-Handle. Dieses Handle wird von bestimmten Blinker-Routinen als Parameter benötigt.

Diese Routinen sind ausführlich in Kapitel 10 "Die Blinker-Funktionen" beschrieben. Die Nummer des Handles kann entweder auf 1 (Voreinstellung) zur Ausgabe auf dem Bildschirm, auf 2 zur Ausgabe auf der Standard-Fehlereinheit (standard error device) oder auf 4 zur Ausgabe zum Drucker gesetzt werden.

Siehe auch: BLIDBGHAN()

BLINKER DEMONSTRATION DATE

Zweck: Gibt das Ablaufdatum für eine Demo-Version an

Syntax: `BLINKER DEMONSTRATION DATE <jjjj/mm/tt>`

Standard: kein

Beispiel:
```
# Setzen des Ablaufdatums
BLINKER DEMONSTRATION DATE 1992/12/31
```

Bemerkung: Dieser Blinker-Befehl stellt quasi das 'Verfallsdatum' einer Applikation ein. Nach diesem Datum kann die Applikation nicht mehr gestartet werden, sondern bricht mit der Meldung ab, daß das Grenzdatum der Demo-Version überschritten worden ist. Es wird im amerikanischen Format jjjj/mm/tt angegeben.

Hiermit können Sie Demo-Versionen Ihrer Applikation erstellen, ohne den Quellcode der Applikation ändern zu müssen. Wenn ein Verfallsdatum gesetzt worden ist und der Overlaymanager nach diesem Datum das erste Overlay lädt (entscheidend ist das System-Datum), dann wird die Fehlerbehandlungs-Prozedur BLERRPRG mit der Fehlernummer 1208 aufgerufen. Diese zeigt an, daß das letzte gültige Aufrufdatum der Demo-Version überschritten worden ist. Der Quellcode dieser Prozedur liegt bei, so daß Sie die Meldung den eigenen Wünschen entsprechend modifizieren können.

Hinweis: Bitte beachten Sie: Die Demo-Restriktionen werden nur überprüft, wenn Overlays geladen werden, also stellen Sie bei ihrem Gebrauch sicher, daß mindestens eine regelmäßig aufgerufene Funktion oder Prozedur im Overlay liegt. Um sicherzustellen, daß Overlays existieren, sollten Sie daran denken, daß alle Segmente, die kleiner als der BLINKER OVERLAY THRESHOLD-Standard von 32 Bytes sind, automatisch aus dem Overlay-Bereich ausgelagert werden.

Anmerkung zu CA-Clipper 5.x:

Damit der Overlay-Manager des Blinker in die Anwendung eingebunden wird, muß etwas nicht-CA-Clipper-Code, wie z.B. EXTEND.LIB, in Overlays gelegt werden.

Siehe auch: BLINKER DEMONSTRATION MINUTES
BLIDEMDTE() / BLIDEMDTEBAS()

BLINKER DEMONSTRATION MINUTES

Zweck: Gibt die Anzahl der Minuten für eine Demo-Version an

Syntax: `BLINKER DEMONSTRATION MINUTES <nuLimit>`

Standard: kein

Beispiel:
```
# Setzen der maximalen Laufzeit des Programms
BLINKER DEMONSTRATION MINUTES 20
```

Bemerkung: Mit diesem Blinker-Befehl können Sie die Laufzeit einer Applikation auf eine bestimmte Anzahl von Minuten begrenzen, nach der dann die Applikation unter Ausgabe der Meldung

'demonstration minutes limit exceeded' abbricht. Es können Werte zwischen 1 und 65535 angegeben werden.

Hiermit können Sie Demo-Versionen Ihrer Applikation erstellen, ohne den Quellcode der Applikation ändern zu müssen. Wenn das angegebene Zeitlimit erreicht ist, wird die Fehlerbehandlungs-Prozedur BLERRPRG mit der Fehlernummer 1209 aufgerufen. Diese zeigt an, daß die maximale Ausführungszeit der Demo-Version überschritten ist.

Der Quellcode dieser Prozedur liegt bei, so daß Sie die Meldung den eigenen Wünschen entsprechend modifizieren können.

Dieser Befehl wird momentan für Dual Mode- oder Extended Mode-Programme nicht unterstützt.

Hinweis: Bitte beachten Sie: Die Demo-Restriktionen werden nur überprüft, wenn Overlays geladen werden, also stellen Sie bei ihrem Gebrauch sicher, daß mindestens eine regelmäßig aufgerufene Funktion oder Prozedur im Overlay liegt. Um sicherzustellen, daß Overlays existieren, sollten Sie daran denken, daß alle Segmente, die kleiner als der BLINKER OVERLAY THRESHOLD-Standard von 32 Bytes sind, automatisch aus dem Overlay-Bereich ausgelagert werden.

Die Funktion BLIDEMMIN() gestattet die Erweiterung des Limits aus dem Programm heraus. Nähere Informationen finden Sie in dem entsprechenden Abschnitt in Kapitel 10.

Anmerkung zu CA-Clipper 5.x :

Damit der Overlay-Manager des Blinker in die Anwendung eingebunden wird, muß ein nicht-CA-Clipper-Code, wie z.B. EXTEND.LIB, ins Overlay abgelegt werden.

Siehe auch: BLINKER DEMONSTRATION DATE
BLIDEMMIN()

BLINKER ENVIRONMENT CLIPPER

Zweck: Gibt den Namen der CA-Clipper-Umgebungsvariablen an

Syntax: `BLINKER ENVIRONMENT CLIPPER <cName>`

Standard: <cName> ist CLIPPER

Beispiel:
```
# Ändern des CLIPPER-Variablennamens in Newclip
BLINKER ENVIRONMENT CLIPPER Newclip
```

Bemerkung: Dieser Befehl erlaubt es dem Entwickler, den Namen der CLIPPER-Umgebungsvariablen von der Standardeinstellung 'CLIPPER' in jeden beliebigen, maximal 16 Zeichen langen String umzuwandeln.

In Verbindung mit dem Befehl BLINKER ENVIRONMENT OVERRIDE erlaubt es dieser Befehl, die durch BLINKER EXECUTABLE CLIPPER 'eingebrannten' Einstellungen zur Laufzeit mit dieser neu erstellten Variablen zu überschreiben. Die Wahl eines neuen Namens vermeidet Konflikte mit schon bestehenden CLIPPER-Umgebungseinstellungen. Dazu ein Beispiel:

```
BLINKER EXECUTABLE CLIPPER F31;X100
BLINKER ENVIRONMENT OVERRIDE
```

```
BLINKER ENVIRONMENT CLIPPER ACCSYS
```

Auch wenn auf DOS-Ebene

```
SET CLIPPER=F21;
```

gesetzt ist, wird diese Variable vom Programm ignoriert werden, und die BLINKER EXECUTABLE CLIPPER-Einstellung kann von folgendem DOS-Befehl überschrieben werden:

```
SET ACCSYS=F99;
```

Siehe auch: BLINKER ENVIRONMENT NAME
BLINKER ENVIRONMENT OVERRIDE
BLINKER EXECUTABLE CLIPPER

BLINKER ENVIRONMENT NAME

Zweck: Gibt den Umgebungsvariablennamen für Blinker-Laufzeitoptionen an

Syntax: `BLINKER ENVIRONMENT NAME <cName>`

Standard: <cName> ist BLINKER

Beispiel:
```
# Umbenennen der Blinker-Variablen
BLINKER ENVIRONMENT NAME Testapp
```

auf dem DOS-Prompt könnte eingegeben werden:

```
SET TESTAPP=/OO30
```

um den OPSIZE-Wert zur Programmlaufzeit auf 30KB einzustellen, ohne neu linken zu müssen

Bemerkung: Dieser Befehl wird benutzt, um den Namen der Umgebungsvariablen BLINKER zu ändern.

Während der Laufzeit einer Applikation sucht der Blinker-Overlay-Manager die DOS-Umgebung nach der Umgebungsvariablen BLINKER ab. Falls diese Variable definiert worden ist, wird sie nach bestimmten Laufzeitparametern durchsucht.

Wenn mit obigem Befehl gelinkt wurde, sucht der Overlay-Manager in der Umgebung nach der Variablen TESTAPP. Ein Listing mit den gültigen Parametern der Umgebungsvariable BLINKER finden Sie im Kapitel 2.

Siehe auch: BLINKER OVERLAY OPSIZE
BLINKER OVERLAY PAGEFRAME
BLINKER OVERLAY UMB

BLINKER ENVIRONMENT OVERRIDE

Zweck: Gibt an, daß die Voreinstellungen durch die CLIPPER Umgebungseinstellungen überschrieben werden dürfen.

Syntax: `BLINKER ENVIRONMENT OVERRIDE`

Standard: Deaktiviert

Bemerkung: In der Voreinstellung können die mit dem Blinker-Befehl BLINKER EXECUTABLE CLIPPER eingestellten Parameter nicht während der Laufzeit der Applikation überschrieben werden. Dadurch können verschiedene Applikationen mit unterschiedlichen Anforderungen auf dem gleichen Rechner ohne Änderung der CLIPPER-Environment-Einstellungen ablaufen.

Dieser Blinker-Befehl bewirkt, daß die mit dem Befehl BLINKER EXECUTABLE CLIPPER eingestellten Parameter zur Laufzeit der Applikation mit den Einstellungen der DOS-Umgebungsvariablen CLIPPER überschrieben werden können. Falls Sie diese Möglichkeit nutzen wollen, so können Sie auch vorher die CLIPPER-Umgebungsvariable mittels des Befehls BLINKER ENVIRONMENT CLIPPER umbenennen.

Siehe auch: BLINKER ENVIRONMENT CLIPPER
BLINKER EXECUTABLE CLIPPER

BLINKER EXECUTABLE ALIGNMENT

Zweck: Legt die Ausrichtungsgröße für eine Windows-.EXE-Datei fest.

Syntax: `BLINKER EXECUTABLE ALIGNMENT <dateiausrichtung>[,<abschnittssausrichtung>]`

Standard: 16-Bit .EXE-Dateien 64 CA-Clipper.
16 Alle anderen Sprachen.

32-Bit .EXE-Dateien 512, 4096.

Beispiel:
```
# Ausrichtung auf 8 Bytes für eine
# kleine .EXE-Datei ändern
BLINKER EXECUTABLE ALIGNMENT 8
```

Bemerkung: Dieser Befehl legt die Ausrichtungsgröße und daher auch die maximale Dateigröße für die neue Datei im .EXE-Format unter dem DOS-Extender sowie für Windows-Programme fest.

Auf Segmente und Ressourcen innerhalb neuer 16-Bit .EXE-Dateien wird über 16-Bit-Seiten-nummern zugegriffen, wodurch die Anzahl der Seiten auf 65535 und die maximale Größe der .EXE-Datei auf 1 MB beschränkt ist.

Bereiche und Ressourcen in portierbaren 32-Bit-.EXE-Dateien werden gemäß der Datei-Ausrichtungs-Einstellung in der ausführbaren Datei und dem festgelegten Abschnitts-Ausrichtungswert beim Laden in den Speicher behandelt

Der Wert für die Ausrichtung kann jede beliebige Potenz von 2 annehmen - je größer der Wert angegeben wird, desto mehr Speicher wird allerdings auch unnötig verbraucht.

Ausrichtung	Max. 16-Bit .EXE-Größe
1	64 KB
2	128 KB
4	256 KB
8	512 KB
16	1 MB
32	2 MB
64	4 MB
128	8 MB

BLINKER EXECUTABLE CLIPPER

Zweck: Gibt die Voreinstellung für die Umgebungsvariable CLIPPER an.

Syntax: `BLINKER EXECUTABLE CLIPPER <cUmgebungs-String>`

Standard: kein

Beispiel: `BLINKER EXECUTABLE CLIPPER F35`

Bemerkung: Dieser Blinker-Befehl dient dazu, Vorgabewerte für die Laufzeit-Umgebungsvariable von CA-Clipper in die ausführbare .EXE-Datei ‘einzubrennen’.

Diese Einstellungen haben Vorrang vor denen, die mit dem SET CLIPPER-Befehl gesetzt wurden. Parameter, die nicht mit dem Befehl BLINKER EXECUTABLE CLIPPER spezifiziert wurden, behalten ihre ursprünglich von SET CLIPPER zugewiesenen Werte. Wenn Sie jedoch den Werten im Befehl SET CLIPPER den Vorrang geben möchten, benutzen Sie den Befehl BLINKER ENVIRONMENT OVERRIDE.

Die maximale String-Länge, die in eine CA-Clipper 5.x-Applikation eingebrannt werden kann, beträgt 128 Bytes, und Blinker ignoriert alle bis auf den LETZTEN angetroffenen 'Einbrenn'-Befehl, sodaß alle benötigten Einstellungen in einer einzigen BLINKER EXECUTABLE CLIPPER Anweisung vorgenommen werden sollten.

BEACHTEN SIE BITTE, DAß DIESER UND ANDERE BLINKER-BEFEHLE NUR DANN WIRKSAM WERDEN, WENN EIN KOMPLETTER (KEIN INKREMENTELLER) LINKLAUF DURCHGEFÜHRT WIRD.

Die eingebrannten Umgebungswerte können nicht mittels der CA-Clipper Funktionen GETE() / GETENV() ermittelt werden.

Anmerkung für CA-Clipper 5.x:

Die hier aufgeführten Laufzeitparameter sind auf alle CA-Clipper 5.x-Programme anwendbar. Entnehmen Sie bitte die komplette Liste dieser Parameter, ihre Syntax und ihre Standardwerte der neuesten CA-Clipper-Dokumentation.

E:nnnn	Expanded Speicher in KB
F:nnn	Anzahl der Dateien
X:nn	Ausschluß von Speicher in KB

BADCACHE Schützen der EMS-Pageframe
CGACURS den EGA-Cursor abschalten
DYNF:nn Anzahl der Handles für das Dynamic Paging
NOIDLE die im Hintergrund ablaufende Speicherbereinigung abschalten
SWAPK:nnnn Größe der Auslagerungsdatei
SWAPPATH:'Pfad' Pfad zur Auslagerungsdatei
TEMPPATH:'Pfad' Pfad zu temporären Dateien

CA-Clipper 5.x-Umgebungseinstellungen können immer auch auf der Kommandozeile der Applikation überschrieben werden (dies sogar ungeachtet der BLINKER ENVIRONMENT OVERRIDE-Einstellung). Dazu wird der Kommandozeilenschalter '//' verwendet, z.B.:

```
MYAPP //SWAPPATH:'D:\'
```

was CA-Clipper dazu veranlassen würde, die VMM-Auslagerungsdatei auf Laufwerk D: in der Root zu erstellen.

Normalerweise besteht kein Grund, die EMS-Unterstützung für CA-Clipper 5.x-Applikationen auszuschalten(E0).

Anmerkung für CA-Clipper Sommer '87:

Die CA-Clipper-Laufzeiteinstellungen werden detailliert in Kapitel 9 - Die Laufzeitumgebung - des CA-Clipper Sommer 87-Benutzerhandbuchs beschrieben.

Sie bestehen aus sechs Werten:

Ennn Expanded Memory in KB
Fnnn Anzahl der offenen Dateien
Rnnn 'Run'-Befehl/Index-Speicher in KB
Sn 'Schnee' abschalten
Vnnn Variablentabelle in KB
Xnnn Ausschluß von Speicher in KB

Um sicherzustellen, daß Ihre Applikation z. B. 41 Dateien gleichzeitig verwalten kann und 100 KB zur Ausführung des RUN-Befehls zur Verfügung stehen, geben Sie bitte folgende Zeile in Ihrer Skriptdatei an:

```
BLINKER EXECUTABLE CLIPPER F41;R100
```

Damit die Applikation auch 41 Dateien öffnen kann, müssen in der CONFIG.SYS und ggf. in der Netzwerkkonfiguration entsprechende Werte eingetragen sein.

Zur Laufzeit werden dann die Parameter F und R der CA-Clipper-Umgebung unter Berücksichtigung der obigen, eingebrannten Werte ignoriert. Alle anderen Parameter werden entsprechend der Einstellung der DOS-Umgebungsvariablen CLIPPER übernommen. Beachten Sie bitte dabei, daß die CLIPPER-Umgebungsvariable auf jeden Fall zur Deaktivierung der Benutzung von Expanded Memory durch CA-Clipper Sommer 87 eingesetzt werden sollte. Sonst könnten, wenn CA-Clipper Sommer 87 diesen Speicher in Verbindung mit bestimmten Expanded Memory-Treibern benutzt, Indexdateien zerstört werden oder Fehler beim RUN-Befehl auftreten. Um CA-Clipper Sommer 87 am Gebrauch von Expanded Memory zu hindern, setzen Sie die CLIPPER-Umgebungsvariable 'E' wie folgt auf 0 (evtl. zusammen mit anderen verwendeten Parametern, z. B Anzahl geöffneter Dateien):

```
SET CLIPPER=E0
```

Alternativ können Sie auch E0 in die Applikation wie folgt ‘einbrennen’:

```
BLINKER EXECUTABLE CLIPPER E0
```

Siehe auch: BLINKER ENVIRONMENT CLIPPER
BLINKER ENVIRONMENT OVERRIDE

BLINKER EXECUTABLE COMPRESS

Zweck: Legt den Komprimierungsgrad für eine .EXE-Datei fest.

Syntax: `BLINKER EXECUTABLE COMPRESS [<faktor>]`

Standard: <faktor> ist 1 (maximale Komprimierung).

Beispiel:

```
# Maximale Komprimierung für die
# vertriebsfertige Version
BLINKER EXECUTABLE COMPRESS 1
```

Bemerkung: Dieser Befehl wird verwendet, um die Kompri-mierung der zu erzeugenden .EXE-Datei einzuschalten und optional auch den Komprimierungsgrad festzulegen. Wird dieser Faktor nicht angegeben, ist sein Vorgabewert 1.

Der Komprimierungsgrad kann jeden Wert zwischen 1 und 9 betragen, wobei 1 die größtmöglichste Komprimierung (und auch die längste Linkzeit) bewirkt und 9 die schnellste Linkzeit mit einer noch vernünftigen Kompressionsrate zur Folge hat.

.EXE- und .DLL-Dateikomprimierung wird im Detail im Abschnitt ‘.EXE- und .DLL-Dateikomprimierung’ im Kapitel 7 dieses Handbuchs beschrieben. Dort finden Sie weitergehende Informationen zu Vorteilen und Auswirkungen der Komprimierung von .EXE-Dateien.
Eine Kompression von VB DOS Programmen, die CHAIN verwenden, wird zur Zeit nicht unterstützt, da die Runtime Bibliothek versucht, Daten aus mit CHAIN verknüpften Dateien zu lesen.

BLINKER EXECUTABLE DUAL

Zweck: Gibt die Erzeugung einer im Dual Mode ausführbaren Datei an.

Syntax: `BLINKER EXECUTABLE DUAL [<minimaler Erweiterungsspeicher>]`

Standard: <minimaler Erweiterungsspeicher> beträgt 512 KB

Beispiel:

```
# Kann im Protected Mode ausgeführt werden, wenn mindestens
# 2 MB zur Verfügung stehen
BLINKER EXECUTABLE DUAL 2048
```

Bemerkung: Dieser Befehl weist an, daß der Blinker eine ausführbare Dual Mode-Datei erzeugen soll, die sowohl im Real als auch im Protected Mode ausgeführt werden kann. Der optionale Parameter dieses Befehls enthält die Minimalgröße des Erweiterungsspeichers, der

verfügbar sein muß, wenn das Programm im Protected Mode ausgeführt werden können soll.

Zur Laufzeit wird das Programm im Real Mode ausgeführt, wenn einer der folgenden Punkte wahr wird:

- Die CPU ist kleiner als 80286.
- Der Prozessor ist ein 80286-er oder höher. Es steht aber kein DPMI-, VCPI- oder XMS-Host zur Verfügung.
- Der Prozessor ist ein 80286-er (oder höher), aber die Größe des freien Erweiterungsspeichers ist kleiner als die angegebene Minimalgröße.

Ansonsten wird das Programm im Protected Mode ausgeführt.

Falls keine BEGINAREA und ENDAREA Befehle mit BLINKER EXECUTABLE DUAL verwendet werden, dann werden ALLE Overlays deaktiviert, und es wird ein Programm ohne Overlays erzeugt, daß sowohl im Real Mode als auch im Protected Mode laufen kann.

Hinweis: Die Programierer von Dual Mode-Programmen müssen bestimmte Spezifika beachten, damit Programme sowohl im Real als auch im Protected Mode arbeiten. Die Verwendung dieses Befehls sichert nicht ab, daß ein Programm auch wirklich erfolgreich als Dual Mode-Programm eingesetzt werden kann. Der Befehl setzt lediglich das Format der .EXE-Datei entsprechend.

Eine detaillierte Beschreibung der Anforderungen eines Dual Mode-Programms finden Sie im Kapitel 5 ‘Programierung von Dual Mode-Anwendungen’.

BLINKER EXECUTABLE EXTENDED

Zweck: Gibt die Erzeugung eines DOS-Extender-Programms an

Syntax: `BLINKER EXECUTABLE EXTENDED <minimaler Erweiterungsspeicher>`

Standard: <minimaler Erweiterungsspeicher> beträgt 512 KB

Beispiel:
```
# Erzeugt ein DOS-Extender-Programm, das 2 MB Erweiterungs
# speicher benötigt
BLINKER EXECUTABLE EXTENDED 2048
```

Bemerkung: Über diesen Befehl wird angegeben, daß der Blinker ein im Protected Mode lauffähiges DOS-Extender-Programm erzeugen soll.

Die erzeugte EXE-Datei kann nur auf einem Rechner mit einem 80286-er (oder höherentwickelten) Prozessor ausgeführt werden, wobei mindestens <minimaler Erweiterungsspeicher> Erweiterungsspeicher zur Verfügung stehen muß.

Wenn BLINKER EXECUTABLE EXTENDED angegeben wird, dann werden ALLE Linkskriptbefehle, die mit Overlays zu tun haben, ignoriert, und es wird immer ein Programm ohne Overlays erzeugt.

Hinweis: In Protected Mode-Programmen müssen bestimmte Auflagen eingehalten werden, damit sie im Protected Mode laufen. Allein die Verwendung dieses Befehls sichert nicht ab, daß ein

Programm auch wirklich erfolgreich als Protected Mode-Programm eingesetzt werden kann. Der Befehl setzt lediglich das Format der .EXE-Datei entsprechend.

BLINKER EXECUTABLE IPX

Zweck: Angabe der Anzahl von IPX ECBs und Sendepuffer.

Syntax: `BLINKER EXECUTABLE IPX <ecb count> [,<send buffer count>]`

Standard: <ecb count> ist 64
<send buffer count> ist 32

Bemerkung: Der Blinker DOS Extender bietet eine transparente Unterstütztung von Novell IPX and SPX (Novell Nezwerk Kommunikation). Dazu müssen vom Extender sowohl zum Senden als auch zum Empfangen Puffer im unteren Speicherbereich reserviert werden.

Dieser Befehl gibt an, wieviele IPX ECBs und Sendepuffer das DOS Extended Programm verwenden kann.

Das IPX/SPX Modul verwendet zwei verschiedene Puffertypen. Der erste Typ sind die ECBs (Event Control Block), von denen jeweils einer pro wartendem Sende- oder Empfangsereignis benötigt wird. Der zweite Puffertyp sind die Sendepaketpuffer, die Daten enthalten, welche schon an IPX/SPX übergeben, aber noch nicht gesendet sind (pending send events). Im Falle der ECBs ist die erforderliche Anzahl gleich der Gesamtzahl der anstehenden Ereignisse, die an IPX weitergegeben, aber zu einem beliebigen Zeitpunkt noch nicht abgeschlossen sind. Die Anzahl der erforderlichen Sendepuffer ist die Gesamtzahl der gleichzeitig aktiven Sendevorgänge.

Jeder angegebene ECB Puffer erfordert etwa 42 Byte konventioneller Speicher, und jeder angegebener Sendepuffer erfordert etwa 550 Byte. Die Gesamtgröße aus beiden Puffertypen zusammen darf 64KB nicht überschreiten.

Wenn Ihr Programm nicht gerade sehr viele IPX Empfangsaufträge vergibt oder sehr viele Pakete gleichzeitig versendet, müßten die Standardwerte ausreichend sein.

Vor dem ersten IPX Aufruf wird vom Extender kein Speicher reserviert.

BLINKER EXECUTABLE NOBLINK

Zweck: Verursacht, daß KEIN Blinker-Code mit hinzugelinkt wird

Syntax: `BLINKER EXECUTABLE NOBLINK`

Standard: Deaktiviert

Bemerkung: Durch diesen Befehl wird Blinker dazu aufgefordert, alle Overlays, Demo-Beschränkungen und das Paging-System von CA-Clipper 5.x zu ignorieren. Weiterhin werden Externals für alle Blinkerfunktionen, einschließlich der Swap-Funktionen nicht aufgelöst. Der Befehl wird hauptsächlich verwendet, um kleine C-/ ASM-Programme ohne Overhead zu erzeugen.

VERWENDEN SIE DEN BEFEHL NICHT FÜR CA-CLIPPER AUSSER BEI KLEINEN PROGRAMMEN OHNE OVERLAYS !!

BLINKER EXECUTABLE NODELETE

Zweck: Erzeugen der .EXE trotz evtl. vorhandener unaufgelöster externer Referenzen

Syntax: `BLINKER EXECUTABLE NODELETE`

Standard: Ausgeschaltet

Bemerkung: Dieser Befehl verhindert, das Blinker .EXE-Dateien löscht, auch dann, wenn 'nicht aufgelöste Externals' vorgefunden wurden. Dies kann im Entwicklungszyklus nützlich sein, wenn noch nicht alle Routinen definiert wurden. Beachten Sie jedoch, daß das Programm abstürzen wird, wenn eine unaufgelöste externe Referenz aufgerufen wird.

Beispiel:
```
# Erzeugen einer .EXE trotz unaufgelöster externer Referenzen
BLINKER EXECUTABLE NODELETE
```

BLINKER EXECUTABLE REAL

Zweck: Gibt an, daß ein Real Mode-Programm erzeugt werden soll

Syntax: `BLINKER EXECUTABLE REAL`

Beispiel:
```
# Erzeugt ein Programm, das nur im Real Mode läuft
BLINKER EXECUTABLE REAL
```

Bemerkung: Dieser Befehl gibt an, daß Blinker ein Real Mode-Programm erzeugen soll. Das entspricht auch der Voreinstellung, wenn kein anderer Typ für die ausführbare Datei angegeben wurde.

Real Mode-Programme können optional dynamische Overlays enthalten, verwenden aber keinenProtected Mode. Sie laufen daher auf allen Systemen mit 80x86-er Prozessoren.

BLINKER EXECUTABLE SERIAL

Zweck: Gibt eine in die .EXE-Datei zu verschlüsselnde Seriennummer an

Syntax: `BLINKER EXECUTABLE SERIAL <cSeriennummer>`

Beispiel: `BLINKER EXECUTABLE SERIAL 001001 5 Blink, Inc.`

Bemerkung: Dieser Blinker-Befehl codiert eine Seriennummer zusammen mit anderen Informationen von insgesamt maximal 50 Zeichen Länge in die .EXE-Datei mit ein. Während der Laufzeit können diese Informationen mit der Funktion BLISERNUM() abgefragt werden.

Diese Informationen werden in der .EXE-Datei verschlüsselt abgelegt und können z. B. eine Serien- oder Lizenznummer verstecken, sodaß sie als eine Art Kopierschutz dienen können. Alle nach dem Befehl BLINKER EXECUTABLE SERIAL in der Skriptdatei in derselben

Zeile folgenden Zeichen (maximal 100) werden in die ausführbare Datei codiert. Zur Laufzeit wird dieselbe Zeichenkette über die Funktion BLISERNUM() zurückgegeben.

Die Seriennummer in obigem Beispiel kann ausgelesen werden, indem der Rückgabestring von BLISERNUM() mit SUBSTR() in entsprechende Teilstrings aufgesplittet wird:

```
TEMP=BLISERNUM()
? "Seriennummer    :",substr(temp,1,8)
? "Benutzerzahl    :", val(substr(temp,10,1))
? "Linzenznehmer    :",substr(temp,12,20)
```

Siehe auch: BLISERNUM() / BLISERNUMBAS()

BLINKER EXTMEM BLOCK

Zweck: Gibt die Blockgröße für die Belegung des Erweiterungsspeichers an

Syntax: `BLINKER EXTMEM BLOCK <nuGröße>`

Standard: 256

Beispiel:
```
# Belegung erfolgt in Blöcken zu 512 KB
BLINKER EXTMEM BLOCK 512
```

Bemerkung: Über diesen Befehl wird die Speicherblockgröße (in KB) angegeben, die vom Extender bei der Belegung von Erweiterungsspeicher verwendet wird. Wird das Programm im Protected Mode ausgeführt, dann belegt der Extender den vom Host zugeteilten Speicher in Blöcken mit der angegebenen Größe und verwaltet diese Blöcke selbst.

Wenn ein DOS-Extender-Programm Speicher anfordert, dann belegt es Speicher in diesen vom Extender und nicht direkt vom Host verwalteten Blöcken. Für Programme, die viel Speicher benötigen, ist eine größere Blockgröße (größer als die standardmäßigen 256 KB) von Nutzen.

Beachten Sie, daß die Einstellung /EBnnnn in der BLINKER Umgebungsvariablen diese Funktion überschreibt.

Siehe auch: BLINKER EXTMEM LIMIT

BLINKER EXTMEM LIMIT

Zweck: Gibt die Gesamtgröße des Erweiterungsspeicher-Bereiches an, der vom Extender belegt wird.

Syntax: `BLINKER EXTMEM LIMIT <nuGröße>`

Standard: Sämtlicher verfügbarer Erweiterungsspeicher

Bemerkung: Dieser Befehl gibt die Gesamtmenge an physischem Speicher (in KB) an, die der Extender belegt, bevor das VM-System des Extenders damit beginnt, bestimmte Daten auf die Festplatte auszulagern (swappen), um weiteren Speicherbedarf zu befriedigen. Über diesen Befehl kann abgesichert werden, daß ein DOS-Extender-Programm nicht notwendigerweise sämtlichen verfügbaren Speicher belegt. Außerdem kann dadurch eine Systemumgebung mit geringerem Speicherangebot simuliert werden.

Beachten Sie, daß die Einstellung /ELnnnn in der BLINKER Umgebungsvariablen diese Funktion überschreibt.

Siehe auch: BLINKER EXTMEM BLOCK

BLINKER HOST DPMI

Zweck: Aktiviert / deaktiviert die Verwendung eines DPMI-Host durch den Extender.

Syntax: `BLINKER HOST DPMI ON/OFF`

Standard: ON (aktiviert die Verwendung des DPMI)

Bemerkung: Mit diesem Befehl kann die Nutzung eines DPMI-Host durch den Extender ausgeschaltet werden, um in einer Umgebung ohne DPMI-Host testen zu können, oder wenn man vermutet, daß der DPMI-Host Probleme verursacht.

Unter normalen Umständen sucht der Extender in der folgenden Reihenfolge nach Hosts:

DPMI
VCPI
XMS

Beachten Sie, daß die Einstellung /HDc in der BLINKER Umgebungsvariablen diese Einstellung überschreibt.

Siehe auch: BLINKER HOST VCPI
BLINKER HOST XMS
BLINKER HOST QDPMI

BLINKER HOST MESSAGE

Zweck: Aktiviert/deaktiviert Statusmeldung beim Programmstart.

Syntax: `BLINKER HOST MESSAGE ON | OFF`

Standard: OFF

Bemerkung: Mit diesem Befehl können bei Start eines Programmes verschiedene Informationen bzgl. der verfügbaren Hosts angezeigt werden.

Dieser Befehl bietet eine einfache Methode, beim Start eines Programmes im Extended oder Dual Mode den aktuellen Host zu überprüfen, sowie die Blinker Versionnsnummer, den Prozessortyp und den verfügbaren Speicher.

Beispiel:

```
BLX286 : EMS/XMS sharing present
BLX286 : DPMI host not detected
BLX286 : XMS host detected
BLX286 : VCPI host detected
BLX286 : DPMI host emulator initialised
BLX286 : Active host is VCPI (v4.10 i486 5668 Kb)
```

Beachten Sie, daß die Einstellung /HDc in der BLINKER Umgebungsvariablen diese Einstellung überschreibt

BLINKER HOST PASSUP

Zweck: Ein-/Ausschalten der automatischen Weitergabe von IRQ-Interrupts

Syntax: `BLINKER HOST PASSUP ON | OFF`

Standard: ON

Bemerkung: Dieser Befehl schaltet die automatische Weitergabe von IRQ-Interrupts vom Real Mode an den Protected Mode durch den Blinker Extender ein oder aus.

Dieser Befehl sollte nur für Debug-Zwecke erforderlich sein, wenn Sie Interrupt-Handler testen, und ist standardmäßig eingeschaltet.

Beachten Sie, daß die Einstellung /HPc in der BLINKER Umgebungsvariablen diese Einstellung überschreibt

BLINKER HOST QDPMI

Zweck: Aktiviert / deaktiviert die Verwendung eines QDPMI-Host durch den Extender.

Syntax: `BLINKER HOST QDPMI ON/OFF`

Standard: ON (aktiviert die Verwendung des QDPMI)

Bemerkung: Mit diesem Befehl kann die Nutzung eines QDPMI-Host durch den Extender ausgeschaltet werden, um in einer Umgebung ohne QDPMI- Host testen zu können, oder wenn man vermutet, daß der QDPMI-Host Probleme verursacht.

Unter normalen Umständen sucht der Extender in der folgenden Reihenfolge nach Hosts:

DPMI
VCPI
XMS

Beachten Sie, daß die Einstellung /HQc in der BLINKER Umgebungsvariablen diese Einstellung überschreibt.

Siehe auch: BLINKER HOST VCPI
BLINKER HOST XMS
BLINKER HOST DPMI

BLINKER HOST VCPI

Zweck: Aktiviert / deaktiviert die Verwendung eines VCPI-Host durch den Extender.

Syntax: `BLINKER HOST VCPI ON/OFF`

Standard: ON (aktiviert die Verwendung des VCPI)

Bemerkung: Mit diesem Befehl kann die Nutzung eines VCPI-Host durch den Extender ausgeschaltet werden, um in einer Umgebung ohne VCPI-Host testen zu können, oder wenn man vermutet, daß der VCPI-Host Probleme verursacht.

Unter normalen Umständen sucht der Extender in der folgenden Reihenfolge nach Hosts:

DPMI
VCPI
XMS

Beachten Sie, daß die Einstellung /HVc in der BLINKER Umgebungsvariablen diese Einstellung überschreibt.

Siehe auch: BLINKER HOST DPMI
BLINKER HOST XMS
BLINKER HOST QDPMI

BLINKER HOST XMS

Zweck: Aktiviert / deaktiviert die Verwendung eines XMS-Host durch den Extender.

Syntax: `BLINKER HOST XMS ON/OFF`

Standard: ON (aktiviert die Verwendung von XMS)

Bemerkung: Mit diesem Befehl kann die Nutzung eines XMS-Host durch den Extender ausgeschaltet werden, um in einer Umgebung ohne XMS-Host testen zu können, oder wenn man vermutet, daß der XMS-Host Probleme verursacht.

Unter normalen Umständen sucht der Extender in der folgenden Reihenfolge nach Hosts:

DPMI
VCPI
XMS

Beachten Sie, daß die Einstellung /HXc in der BLINKER Umgebungsvariablen diese Einstellung überschreibt.

Siehe auch: BLINKER HOST DPMI
BLINKER HOST VCPI
BLINKER HOST QDPMI

BLINKER INCREMENTAL FILE

Zweck: Gibt einen alternativen Dateinamen für die inkrementelle Datei an

Syntax: `BLINKER INCREMENTAL FILE [L:][Pfad]<Dateiname>`

Standard: <Exename>.BIF

Beispiel:
```
# Erzeugen der.BIF-Datei auf RAM-Laufwerk E:
BLINKER INCREMENTAL FILE E:Temp.BIF
```

Bemerkung: Dieser Blinker-Befehl gibt einen alternativen Dateinamen für die Inkremental-Datei der betreffenden Applikation an. Als Vorgabename wird der Name der .EXE-Datei mit der Erweiterung .BIF verwendet. Bitte überprüfen Sie, ob der neu gewählte Dateiname nicht schon als Inkremental-Datei für eine andere Applikation vorhanden ist.

Siehe auch: BLINKER INCREMENTAL OFF
BLINKER INCREMENTAL PAD

BLINKER INCREMENTAL OFF

Zweck: Schaltet den inkrementellen Link-Modus aus

Syntax: `BLINKER INCREMENTAL OFF | ON`

Standard: ON

Bemerkung: Dieser Blinker-Befehl schaltet den (voreingestellten) inkrementellen Link-Modus aus. Das hat zur Folge, daß jedes Linken einen kompletten Linklauf darstellt, keine .BIF-Datei erzeugt und kein Reservespeicher zwischen den einzelnen Prozeduren eingerichtet wird, um inkrementelles Linken zu ermöglichen.

Dieser Befehl sollte auf jeden Fall für die endgültige Version der Applikation verwendet werden, um die bestmögliche Speicherausnutzung zu erhalten.

Komprimierung der Symboltabelle

Wird dieser Befehl eingesetzt, wird dadurch ebenso erreicht, daß die Symboltabellen sowohl für Sommer '87, als auch für CA-Clipper 5.x komprimiert werden. Dies verringert die Größe der Applikations-Symboltabelle, indem doppelte Symbole entfernt werden. Die Größe der Reduzierung hängt zum einen vom Programmierstil, zum anderen von der Größe der Applikation ab.

Dynamic Paging

Wird dieser Befehl für CA-Clipper 5.x eingesetzt, so schalten Sie ebenfalls auch das Dynamic Paging des CA-Clipper Codes ein, was zu einer Steigerung der Laufzeitgeschwindigkeit für CA-Clipper-Code im Overlay führt, ohne den Speicher dabei über die Maßen zu beanspruchen.

Siehe auch: BLINKER INCREMENTAL FILE
BLINKER INCREMENTAL PAD
BLINKER CLIPPER PAGE

BLINKER INCREMENTAL PAD

Zweck: Gibt die Größe des Reservespeichers beim inkrementellen Linken an

Syntax: `BLINKER INCREMENTAL PAD <nuGröße>`

Standard: <nuGröße> ist 128 Bytes

Beispiel:
```
# füge 200 Bytes an das Ende jeder Prozedur an
BLINKER INCREMENTAL PAD 200
```

Bemerkung: Mit diesem Blinker-Befehl können Sie die Größe des Reservespeichers einstellen, der an jede Prozedur angehängt wird, wenn Sie erstmals inkrementell gelinkt haben. Der Parameter ist eine Dezimalzahl und gibt die Anzahl der Bytes an, die an das Ende jeder Prozedur angehängt wird, um Platz für Erweiterungen zu schaffen, und darf jeden beliebigen Wert bis 1024 annehmen. (Der Vorgabewert ist 128, der einen angemessenen Reservespeicher für die

meisten einfachen Änderungen während der Entwicklung einer Routine schafft).

Wenn Sie diesen Wert vergrößern, kann die Applikation häufig inkrementell gelinkt werden und ein komplettes Linken muß nicht allzu oft erfolgen, aber der für die Real Mode-Anwendung in der Testphase zur Laufzeit zur Verfügung stehende Speicher wird dadurch reduziert.

Siehe auch: BLINKER INCREMENTAL FILE
BLINKER INCREMENTAL OFF

BLINKER LINK EMS

Zweck: Gibt an, ob für das Linken EMS Speicher zu verwenden ist

Syntax: `BLINKER LINK EMS ON|OFF`

Standard: ON

Beispiel:
```
# EMS-Verwendung ausschalten, um einen
 # Konflikt mit einem TSR zu verhindern
 BLINKER LINK EMS OFF
```

Bemerkung: Dieser Befehl wird benutzt um festzulegen, ob während des Linklaufs EMS-Speicher durch das Virtual Memory-System des Blinker zu verwenden ist oder nicht. Es gibt normalerweise keinen Grund, diesen Befehl zu verwenden, solange kein Konflikt mit anderer Software zu erwarten ist.

Dieser Befehl setzt automatisch auch BLINKER LINK PAGEFRAME OFF.

Siehe auch: BLINKER LINK PAGEFRAME
BLINKER LINK XMS

BLINKER LINK PAGEFRAME

Zweck: Gibt an, ob die EMS-Pageframe während des Linkens verwendet werden soll

Syntax: `BLINKER LINK PAGEFRAME ON|OFF`

Standard: ON

Beispiel:
```
# Pageframe verbieten, um eventuellen Konflikt
 # mit einem Cache-Programm zu verhindern
 BLINKER LINK PAGEFRAME OFF
```

Bemerkung: Dieser Befehl wird benutzt um festzulegen, ob während des Linklaufs die EMS-Pageframe durch das Virtual Memory-System des Blinker zu verwenden ist, oder nicht. Es gibt normalerweise keinen Grund, diesen Befehl zu verwenden, solange keine Konflikte mit anderer, mitlaufender Software erwartet werden.

Siehe auch: BLINKER LINK EMS

BLINKER LINK VERSION

Zweck: Gibt die niedrigste erforderliche Blinker-Versionsnummer an

Syntax: `BLINKER LINK VERSION <n.n>`

Standard: 0.0

Beispiel:
```
# Blinker Version 2.0 wird benötigt wird, um das
 # Skript zu bearbeiten.
 BLINKER LINK VERSION 2.0
```

Bemerkung: Dieser Befehl wird verwendet, um die niedrigstmögliche Blinker-Version festzulegen, die gebraucht wird, um das Link-Skript zu bearbeiten. Ist die verwendete Version des Blinker kleiner, als in dieser Zeile angegeben, so verweigert Blinker seinen Dienst und bricht den Link-Vorgang mit einer Fehlermeldung ab. Blinker-Versionen vor 2.0 können diesen Befehl nicht auswerten!

BLINKER LINK XMS

Zweck: Gibt an, ob für das Linken XMS-Speicher verwendet werden kann

Syntax: `BLINKER LINK XMS ON|OFF`

Standard: ON

Beispiel:
```
#XMS-Verwendung ausschalten
 BLINKER LINK XMS OFF
```

Bemerkung: Dieser Befehl wird benutzt um festzulegen, ob während des Linklaufs XMS-Speicher durch das Virtual Memory-System des Blinker zu verwenden ist, oder nicht. Es gibt normalerweise keinen Grund, diesen Befehl zu verwenden, solange keine Konflikte mit anderer, mitlaufender Software erwartet werden.

Siehe auch: BLINKER LINK EMS

BLINKER LOWMEM RESERVE

Konventionellen Speicher für Programme reservieren.

Syntax: `BLINKER LOWMEM RESERVE <nuMemSize>`

Standard: <nuMemSize> ist16 KB.

Bemerkung: Dieser Befehl reserviert die angegebene Menge von konventionellem Speicher für das

Programm und verhindert, daß der Extender diesen Speicher für andere Zwecke verwendet.

Unter normalen Umständen benutzt der Extender zunächst den verfügbaren erweiterten Speicher und dann den verfügbaren konventionallen Speicher, um die Speicheranforderungen des Programms zu erfüllen. Dies kann dazu führen, daß kein konventioneller Speicher mehr frei ist für kritische Routinen, die unbedingt konventionellen Speicher benötigen, um mit TSRs, Real Mode Interrupt Behandlungsroutinen oder sonstigem Real Mode Code zu kommubnizieren.

Der mit diesem Befehl reservierte Speicher kann vom Programm mit DOSAllocRealSeg() oder der DPMI Funktion 0x100 alloziert werden.

Einige Third-Party-Bibliotheken können eventuell eine BLINKER LOWMEM RESERVE Einstellung erfordern, um fehlerfrei zu funktionieren. Sehen Sie dazu in der entsprechenden Dokumentation nach oder befragen Sie Ihren Händler für weitere Informationen.

Beispiel:
```
# 32 KB vom konventionellen Speicher reservieren
BLINKER LOWMEM RESERVE 32
```

BLINKER MEMORY CLEAR

Zweck: Initialisiert sämtlichen Speicher jenseits der EXE-Datei auf einen bestimmten Wert

Syntax: `BLINKER MEMORY CLEAR <nuWert>`

Standard: Deaktiviert

Beispiel:
```
# Überschreibt den Speicher mit NULLen
BLINKER MEMORY CLEAR 0
```

Bemerkung: Dieser Befehl bewirkt, daß Blinker sämtlichen Speicher oberhalb des Stack bis zum oberen Speicherende mit dem angegebenen Dezimalwert (0 - 255) initialisiert. Das kann hilfreich sein, um Probleme mit nichtinitialisiertem Speicher zu erkennen etc. Programme sollten normalerweise nicht mit dieser Option ausgeliefert werden. Der Speicher wird initialisiert, bevor die Ausführung des Programms beginnt.

BLINKER MEMORY PACK

Zweck: Gibt die Frequenz für das automatische Packen des Speichers an für CA-Clipper Sommer ‘87

Syntax: `BLINKER MEMORY PACK <nuFrequenz>`

Standard: Deaktiviert

Beispiel:
```
# starte das Packen nach jedem 20-ten
 # Prozedur-Aufruf
 BLINKER MEMORY PACK 20
```

Bemerkung: Dieser Blinker-Befehl stellt die Häufigkeit ein, mit welcher der Blinker-Overlay-Manager den freien Speicher von CA-Clipper Sommer ‘87 packt (d.h. mehrere kleine Speicherfragmente zu einem großen Fragment zusammenfaßt). Der übergebene Parameter ist eine Dezimalzahl zwischen 0 und 65534, die angibt, nach wieviel Prozeduraufrufen der Speicher jeweils gepackt wird. Der Vorgabewert ist 0, das heißt, es erfolgt kein Packen.

CA-Clipper belegt fortlaufend dynamischen Speicher bzw. gibt diesen wieder frei. Dies bewirkt mit zunehmender Programmdauer eine immer größere Speicherfragmentierung. CA-Clipper erkennt nur freie Speicherbereiche, die 4KB oder größer sind, sodaß große Anwendungen nach längerer Ausführung plötzlich aufgrund von Speicherknappheit abbrechen können.

Der Befehl BLINKER MEMORY PACK ermöglicht dem Entwickler einer CA-Clipper Sommer '87-Applikation, den dynamischen Speicher so oft wie nötig zu packen, ohne die Applikation modifizieren zu müssen. Die eingestellte Frequenz kann der jeweiligen Applikation angepaßt werden. Die CA-Clipper-Funktion MEMORY(0) führt den gleichen Vorgang aus, allerdings viel langsamer.

Die Defragmentierung erledigt zwei Dinge: Verbindung von zusammenhängenden freien Speicherblöcken und Zurücksetzung des 'vagabundierenden' Zeigers, der die Aufgabe hat, die nächste Speicherzuweisung auszuwählen (anzuzeigen). Zum Zeitpunkt der Defragmentierung wird dieser Zeiger gesetzt, um den höchsten verfügbaren Ort im Speicher zu spezifizieren. Dadurch wird der Speicher jedesmal vom Anfang an belegt und nicht der ganze Speicherbereich fragmentiert, wodurch die fragmentationsbedingte Verschlechterung des verfügbaren Speichers bedeutend reduziert wird.

Wegen der CA-Clipper Sommer ’87 zugrunde liegenden Architektur der Speicherzuweisungsmethode ist es nicht möglich, belegte Speicherblöcke herumzuschieben, weil es keine Möglichkeit gibt, die/den Zeiger auf die belegten Blöcke zu aktualisieren. Daher ist eine echte garbage collection (Verschiebung von Speicherblöcken zur Erzeugung eines einzigen, zusammenhängenden, freien Speicherblocks) unmöglich, aber die von BLINKER angewandte Defragmentierung ist die zweitbeste Methode und verringert das Problem beträchtlich.

Wenn Sie für nFrequenz eine kleine Zahl angeben, kann die Applikation langsamer ablaufen, während die Fragmentierung, bei einem Maximum an Speichereinsparung auf ein Minimum beschränkt wird. Wenn Sie für nFrequenz einen hohen Wert einsetzen, kommt es zu keiner merklichen Verzögerung der Applikation, aber der Zugewinn an Speicher ist natürlich geringer, weil die Fragmentierung stärker ist (wenn auch immer noch viel günstiger als ohne Packen).

Ein optimaler Wert für eine durchschnittliche Applikation liegt bei 10, wobei ein merklicher Zugewinn an Speicherplatz bei unbedeutender Verlangsamung des Ablaufs erreicht wird. Die optimale Pack-Häufigkeit für eine konkrete Applikation kann erheblich von dieser Frequenz abweichen und ist nur im Trial-and-Error-Verfahren zu ermitteln.

Siehe auch: BLIMEMPAK()
BLIDISFRG()

BLINKER MESSAGE DUPLICATES

Zweck: Schaltet Warnmeldungen für doppelt definierte Symbole innerhalb der Bibliotheken ein

Syntax: `BLINKER MESSAGE DUPLICATES`

Standard: Ausgeschaltet.

Bemerkung: Dieser Befehl erzeugt dann die Warnung 1004, wenn doppelte Symbole in den Bibliotheken während der Linkzeit entdeckt werden.

Dieser Befehl kann als Debugging-Werkzeug sehr hilfreich sein, um doppelte Module in den beim Linken eines Programms benutzten Dateien und Bibliotheken zu identifizieren. Das erste Modul, das in der Meldung 1004 aufgelistet wird, ist das tatsächlich gelinkte Modul. Das zweite aufgelistete Modul ist das Duplikat und wird ignoriert, sofern es nicht mit dem Befehl MODULE FROM überschrieben wird.

Die folgenden Module (und eingeschlossenen Symbole) werden während der Linkzeit bearbeitet und in der ersten Hälfte der VERBOSE-Ausgabe angezeigt. Jedes dieser Symbole wird in der Meldung 1004 aufgelistet, sollte ein anderes bearbeitetes Symbol den gleichen Namen haben.

- Alle im FILE-Befehl angegebenen Module
- Module, die zum Lösen externer Referenzen aus Bibliotheken extrahiert wurden
- Alle Module in einer durch SEARCH spezifizierten Bibliothek

Beachten Sie bitte, daß es mehrere doppelte Module/Symbole in den einbezogenen Bibliotheken geben kann, welche nicht gemeldet werden, wenn BLINKER MESSAGE DUPLICATES aktiviert ist, solange eines oder beide Module nicht bearbeitet werden.

Um den optimalen Nutzen aus diesem Befehl zu ziehen, ist es hilfreich, den Abschnitt ‘Einzelheiten des Link-Prozesses’ in Kapitel 3 zu studieren, wo die von Blinker verwandte Methode zur Bearbeitung von Dateien und Bibliotheken in einem Linklauf abgehandelt wird. Diese Methode entscheidet, welche Module zu bearbeiten sind und welche Module in der vom Befehl BLINKER MESSAGE DUPLICATES erzeugten Ausgabe als Duplikate identifiziert werden.

Um eine vollständige Liste aller für die Applikation maßgeblichen doppelten Module zu bekommen, benutzen Sie diesen Befehl bitte in Verbindung mit dem VERBOSE-Befehl:

(1) Alle möglichen doppelt definierten Symbole auflisten:

Die erste Hälfte der VERBOSE-Ausgabe weist alle während des Linkens bearbeiteten Module aus. Die Duplikate unter diesen Modulen werden vom Befehl BLINKER MESSAGE DUPLICATES gemeldet. Sie können die Bearbeitung aller in einer Bibliothek enthaltenen Module erzwingen, indem Sie die Bibliothek mit dem SEARCH-Befehl bearbeiten. Wenn Sie alle beteiligten Bibliotheken so behandeln, erhalten Sie eine Liste aller Duplikate. Ein Linkvorgang mit all diesen SEARCH-Befehlen dient nur zur Erstellung einer solchen Liste, eine brauchbare .EXE-Datei entsteht so nicht.

(2) Nur die vom Programm benötigten Module identifizieren:

Die zweite Hälfte der VERBOSE-Ausgabe weist nur die tatsächlich von der Applikation verwandten Module aus. Die als Duplikate hervorgehobenen Symbole, deren Module in der zweiten Hälfte der VERBOSE-Ausgabe erscheinen, müssen untersucht werden.

Normalerweise sind alle Duplikate in 'Third-Party'-Bibliotheken oder Dateien desselben Herstellers beabsichtigt und können ignoriert werden; dasselbe gilt für Duplikate in 'Third-Party'-Produkten und Compiler-Bibliotheken.

BLINKER MESSAGE NOBLINK

Zweck: Schaltet die Anzeige der zwinkernden Augen ab

Syntax: `BLINKER MESSAGE NOBLINK`

Standard: Ausgeschaltet.

Bemerkung: Dieser Befehl verhindert die Anzeige der zwinkernden Augen während des Link-Prozesses

Siehe auch: BLINKER MESSAGE WINK

BLINKER MESSAGE NOWARNING

Zweck: Unterdrückt die Ausgabe von Warnungen während des Linkens

Syntax: `BLINKER MESSAGE NOWARNING`

Standard: Ausgeschaltet

Bemerkung: Dieser Befehl verhindert die Ausgabe der Blinker-Warnungen 1001 bis 1100.

BLINKER MESSAGE WINK

Zweck: Schaltet einäugiges Zwinkern während des Linkens ein

Syntax: `BLINKER MESSAGE WINK [LEFT]`

Standard: Rechtes Auge zwinkert

Beispiel:
```
# ein zwinkerndes rechtes Auge darstellen
BLINKER MESSAGE WINK
```

Bemerkung: Dieser Blinker-Befehl aktiviert einäugiges Zwinkern während des Link-Prozesses. Wird kein Parameter angegeben, so zwinkert nur das rechte Auge (aus Blinker's Sicht). Wird als Parameter LEFT angegeben, zwinkert entsprechend das linke Auge.

Siehe auch: BLINKER MESSAGE NOBLINK

BLINKER OVERLAY DYNAMIC

Zweck: Spezifizieren der Methode der dynamischen Overlay-Speicherbelegung für CA-Clipper Sommer ‘87

Syntax: `BLINKER OVERLAY DYNAMIC`

Standard: Eingeschaltet für CA-Clipper Sommer '87

Bemerkung: Dieser Befehl veranlaßt Blinker, dynamisch Overlay-Speicher vom CA-Clipper-Speicher-Pool nach Bedarf zu belegen. Dies ist die Standardmethode der Speicherbelegung für alle Overlay-Applikationen unter CA-Clipper Sommer '87 und die einzige Methode, die Blinker bis zur Version 1.31 einschließlich praktizierte.

Der Vorteil dieses Verfahrens ist, daß Speicher, der durch inaktive Overlays belegt wird, CA-Clipper unter Einsatz der Funktion BLIOVLCLR(), die inaktive Overlays entfernt, wieder zur Verfügung gestellt werden kann. Der Nachteil jedoch besteht darin, daß kein Gebrauch von UMB’s oder der EMS-Pageframe gemacht werden kann, um Overlays ablaufen zu lassen. So kann es passieren, daß Ihnen weniger Speicher für Ihre Applikation zur Verfügung steht.

Siehe auch: BLINKER OVERLAY FIXED
BLIOVLCLR()

BLINKER OVERLAY FIXED

Zweck: Spezifizieren der festen Overlay-Speicherbelegung

Syntax: `BLINKER OVERLAY FIXED`

Standard: Eingeschaltet (optional für CA-Clipper Sommer ’87)

Bemerkung: Dieser Befehl weist Blinker an, beim Programmstart einen Speicherblock fest definierter Größe von DOS für Overlays zu belegen. Ist dieser Speicherblock einmal belegt, wird der Blinker-Overlay-Manager keinen weiteren Speicher mehr für Overlays anfordern. Darum ist der Aufwand für die Overlay-Verwaltung der Applikation vorhersehbar. Diese Methode ermöglicht es, ‘Eltern-Overlays’ (Prozeduren oder Funktionen, welche die aktuelle Prozedur aufgerufen haben) bei Bedarf auszulagern, um Platz für neu auszuführende Prozeduren zu schaffen.

Diese Methode ist die einzige Belegungsmöglichkeit für Overlay-Speicher, die für alle Compiler außer CA-Clipper Sommer ’87 gilt, wo dies optional ist.

Lesen Sie bitte auch den vorherigen Abschnitt, BLINKER OVERLAY DYNAMIC, um die FIXED- und die DYNAMIC- Methode vergleichen zu können.

Siehe auch: BLINKER OVERLAY DYNAMIC

BLINKER OVERLAY OPSIZE

Zweck: Stellt die vorgegebene Arbeitsgröße des Overlay-Pools ein

Syntax: `BLINKER OVERLAY OPSIZE <nuGröße>`

Standard: <nuGröße> ist 40 KB

Beispiel:
```
# Erhöhen des Overlay-Speichers auf 60KB
BLINKER OVERLAY OPSIZE 60
```

Bemerkung: Dieser Befehl bestimmt die Arbeitsgröße des Overlay-Speichers während der Laufzeit in KB. Es können Werte zwischen 12 und 128 eingegeben werden. Die vorgegebene Größe ist 40KB. Sie kann aber auf einen kleineren Wert zum Zugewinn von Speicher oder auf einen größeren Wert zur Verringerung der Plattenzugriffe beim Nachladen von Overlay-Prozeduren eingestellt werden.

Wenn Overlays in einer Skript-Datei spezifiziert werden, ermittelt Blinker die Größe des kleinstmöglichen Overlay-Speichers durch Addition von etwa acht KB zu der Größe der größten Overlay-Einheit (für die meisten Compiler ist dies ein Segment). Dieser Minimalwert wird zusammen mit dem in BLINKER OVERLAY OPSIZE angegebenen Wert angezeigt . Wenn die minimale Größe größer ist als der angegebene Wert, so überschreibt der minimale Wert den angegebenen.

Die Arbeitsgröße des Overlay-Speichers kann nachträglich noch durch die Umgebungsvariable BLINKER eingestellt werden (Kapitel 2, ‘Umgebungsvariablen’). Um zu sehen, wie der Overlay-Speicher während der Laufzeit verwendet wird, schlagen Sie bitte in Kapitel 4 unter dem Abschnitt ‘Overlay-Pool’ nach.

Anmerkung für CA-Clipper Sommer ’87:

Wird CA-Clipper Sommer ’87 zusammen mit der Overlay-Speicherbelegungsmethode DYNAMIC verwendet (siehe BLINKER OVERLAY DYNAMIC), wird die minimale OPSIZE nicht angezeigt, da Sie zur Linkzeit nicht zu ermitteln ist. Die tatsächlich während der Ausführung benötigte OPSIZE kann ermittelt werden, indem zunächst ein kleiner Wert (z.B. 10 KB) angegeben wird. Nun führen Sie alle Punkte Ihrer Applikation aus, o h n e dabei die Funktion BLIOVLCLR() aufzurufen. Direkt vor Beendigung der Applikation und der Rückkehr ins DOS fragen Sie mit der Funktion BLIOVLOPS() die Arbeitsgröße ab. Der von BLIOVLOPS() zurückgegebene Wert ist der minimale Wert für OPSIZE.

Der Grund dafür ist, daß mit der DYNAMIC-Methode alle aktiven Overlays zu einer gegebenen Zeit im Speicher sein müssen. Die minimale Overlay-OPSIZE ist daher die Summe aller aktiven Overlays zu dem Zeitpunkt, da sich das Programm in seiner größten Verschachtelungstiefe befindet. Die oben beschriebene Methode ist ein Weg, die richtige Größe zu ermitteln, und die Wahl eines kleineren Wertes führt einfach dazu, daß er zur Laufzeit überschrieben wird.

Siehe auch: BLINKER ENVIRONMENT NAME
BLIOVLOPS()

BLINKER OVERLAY PAGEFRAME

Zweck: Erlaubt die Benutzung der EMS-Pageframe für Overlays

Syntax: `BLINKER OVERLAY PAGEFRAME ON|OFF`

Standard: OFF

Beispiel:
```
# EMS-Pageframe verwenden
BLINKER OVERLAY PAGEFRAME ON
```

Bemerkung: ACHTUNG: Lesen Sie diesen Abschnitt komplett, bevor Sie diesen Befehl verwenden.

Dieser Befehl erlaubt dem Overlay-Manager, die EMS-Pageframe für den Overlay-Speicher zu verwenden (maximale Größe 64KB), sofern der Rechner mit einem LIM EMS (3.2 oder höher) Expanded-Memory-Board oder Emulatoren ausgestattet ist.

Hinweis: Dieser Befehl wirdzur Laufzeit ignoriert, falls kein EMS-Treiber (oder Pageframe) vorhanden ist.

Um diese Option zu nutzen, müssen mindestens 64 KB EMS-Speicher verfügbar sein und die Overlay-OPSIZE des Programms muß kleiner als 64 KB sein. Während der Laufzeit wird, falls Expanded Memory zur Verfügung steht und diese Option eingeschaltet ist, der Overlay-Manager vier EMS-Seiten belegen, in die Pageframe kopieren und die Pageframe für die Overlay-Ausführung nutzen. Die Overlay-OPSIZE wird automatisch auf 64KB gesetzt, was sich durch eine höhere Ablaufgeschwindigkeit der Applikation bemerkbar macht.

WARNUNG: Wenn dieser Befehl benutzt wird, darf das Programm KEINE Funktionen nutzen, die über die EMS-Pageframe auf das Expanded Memory zugreifen, ES SEI DENN, der Pageframe-Status wurde gesichert und Sie stellen sicher, daß keine Overlay-Funktionen aufgerufen werden, bevor dieser Status wiederhergestellt wurde. Zusätzlich können Funktionen im Overlay nicht über die EMS-Pageframe auf das Expanded Memory zugreifen, ES SEI DENN, sie achten sorgfältig darauf, in welchen EMS-Seiten sie ausgeführt werden. Anderenfalls kann es passieren, daß sie sich selbst aus dem Speicher entfernen. Verstöße gegen diese Spielregeln resultieren in einem Versagen des Programms, einer Verfälschung seines Expanded Memorys, oder in beidem. Beachten Sie auch, daß manche TSR-Programme, Netzwerktreiber und Festplattencache-Programme den Pageframe-Status nicht korrekt sichern bzw. wiederherstellen. Dies kann zu Programmabstürzen führen, wenn Overlays in der Pageframe ausgeführt werden. Die Verwendung der Pageframe kann noch während der Laufzeit durch die BLINKER Umgebungsvariable eingestellt werden (siehe ‘Umbegungsvariablen’, Kap.2).

Wenn dieser Befehl zusammen mit dem UMB-Befehl benutzt wird und sowohl die Pageframe als auch UMB’s verfügbar sind, so wird Blinker sich für die Pageframe entscheiden.

Wird eines oder werden beide spezifiziert und sind auf dem System des Endanwenders nicht verfügbar, so ignoriert Blinker die Anweisungen automatisch und verwendet konventionellen Speicher, also Speicher unter 640 KB.

Anmerkung für CA-Clipper Sommer ’87:

Dieser Befehl ist nur aktiv, wenn mittels BLINKER OVERLAY FIXED die 'Fixed Overlay'-Speicherbelegungsmethode gewählt wurde.

Wird dieser Befehl mit CA-Clipper Sommer ’87 aktiviert, verbietet Blinker die Verwendung von EMS, indem die Einstellung E0 in die CLIPPER-Umgebungsvariable eingebrannt wird.

Anmerkung für CA-Clipper 5.x:

Wird dieser Befehl unter CA-Clipper 5.x eingesetzt, so brennt Blinker automatisch die Einstellung BADCACHE in die CLIPPER-Umgebungsvariable ein, was CA-Clipper 5.x veranlaßt, bei jedem Zugriff auf das Expanded Memory den Pageframe-Status zu sichern. Dies schützt den Overlay-Bereich in der Pageframe vor Defekten.

Siehe auch: BLINKER ENVIRONMENT NAME
BLINKER OVERLAY UMB
BLINKER OVERLAY FIXED

BLINKER OVERLAY THRESHOLD

Zweck: Setzt die Größe des kleinsten Overlay-Segments fest

Syntax: `BLINKER OVERLAY THRESHOLD <nuGrenze>`

Standard: <nuGrenze> ist 32 Bytes

Beispiel:
```
# nur Segmente ins Overlay, die größer sind als 64 Bytes
BLINKER OVERLAY THRESHOLD 64
```

Bemerkung: Dieser Befehl setzt die Schwellengröße fest, unter der fürs Overlay angegebener Code wieder ins Root mit übernommen wird. Sehr kleine Routinen (kleiner als 32 Byte) belegen mehr Platz im Root, wenn sie ins Overlay verfrachtet werden (aufgrund der Tabelleneinträge für den Overlay-Manager), als wenn sie einfach im Root verbleiben.

Darüberhinaus werden sehr kleine Overlay-Routinen unverhältnismäßig langsamer ausgeführt, da die Lade-/Aufrufzeit im Vergleich zur reinen Ausführungszeit des Codes immens sein kann. Falls es der Fall sein sollte, daß viele kleine Routinen sehr häufig aufgerufen werden, liegt es wegen der Verbesserung der Programm-Performance im Interesse des Programmierers, einen Schwellenwert zu spezifizieren, der sie zurück ins Root drängt.

Hinweis für CA-Clipper 5.x: Beachten Sie bitte, daß dieser Befehl keine Auswirkungen auf CA-Clipper 5.x-Code hat, der durch das Paging-Verfahren ins Overlay gelegt wird. Er könnte aber für C-Funktionen innerhalb eines CA-Clipper Programms benutzt werden.

BLINKER OVERLAY UMB

Zweck: Erlaubt die Verwendung von XMS-UMB's für Overlays

Syntax: `BLINKER OVERLAY UMB ON | OFF`

Standard: ON

Beispiel:
```
# XMS-UMB's für den Overlay-Bereich verbieten
BLINKER OVERLAY UMB OFF
```

Bemerkung: Dieser Befehl ermöglicht es dem Blinker Overlay-Manager bei Verwendung eines geeigneten XMS-Treibers, Overlays in Upper Memory Blocks im Bereich über 640 KB auszuführen. Die Verwendung dieser UMB's durch Blinker ist vom Vorhandensein eines zusammenhängenden Blockes ausreichender Größe abhängig. Die Größe des belegten UMB wird mit der Overlay-Arbeitsgröße des Programms identisch sein.

Die Verwendung dieses Befehls erfordert einen 386-Prozessor (oder besser) und einen XMS 2.0 Extended-Memory-Treiber, der UMB's (Upper Memory Blocks) unterstützt. UMB's sind Speicherblöcke, die unter Verwendung der Memory-Management-Fähigkeiten der 386/486 Prozessorfamilie in den Bereich über 640 KB gelegt werden. UMB's werden manchmal auch als 'high memory' oder LOADHI-Bereiche bezeichnet.

Wenn Ihr XMS-Treiber das Laden von Treibern in das 'high memory' unterstützt, werden Sie diese Möglichkeit wohl auch nutzen können. Blinker benötigt jedoch einen UMB, der mindestens so groß ist wie der Overlay-Bereich. Dies kann zur Folge haben, daß Sie die Plazierung Ihrer Programme in diesem Bereich ändern müssen. Die meisten 386-Speicher-Manager verfügen über Utilities, die Sie bei der Optimierung dieses Bereichs unterstützen.

Wird sowohl dieser Befehl als auch der PAGEFRAME-Befehl angegeben und UMB und Pageframe sind auch verfügbar, so wird Blinker sich für die PAGEFRAME entscheiden.

Wenn eine oder beide dieser Möglichkeiten angegeben, aber nicht verfügbar sind, so wird Blinker diese Befehle automatisch ignorieren und den Speicher für den Overlay-Bereich im konventionellen Speicher anfordern. Die Verwendung von UMB's kann noch während der Laufzeit angepaßt werden, indem die BLINKER-Umgebungsvariable dementsprechend geändert wird. (siehe auch 'Umgebungsvariable' in Kapitel 2).

Anmerkung für CA-Clipper Sommer '87:

Dieser Befehl wirkt nur dann, wenn die FIXED- OVERLAY-Belegungsmethode verwendet wird mit dem Befehl BLINKER OVERLAY FIXED.

Siehe auch: BLINKER ENVIRONMENT NAME
BLINKER OVERLAY FIXED
BLINKER OVERLAY PAGEFRAME

BLINKER PROCEDURE DEPTH

Zweck: Gibt die maximale Verschachtelungstiefe von CA-Clipper-Prozeduren an.

Syntax:
```
BLINKER PROCEDURE DEPTH <nuTiefe>
```

Standard: <nuTiefe> ist 50

Beispiel:
```
# den Stack für Fremd-Bibliotheken erhöhen
BLINKER PROCEDURE DEPTH 60
```

Bemerkung: Dieser Befehl wird CA-Clipper-Programmierern als Annehmlichkeit zur Verfügung gestellt und erlaubt die Einstellung des Programmstacks auf der Grundlage der maximalen Verschachtelungstiefe, die während der Ausführung der Applikation auftreten kann. Diese Tiefe entspricht der Anzahl der Prozeduraufrufe, die ausgeführt werden, bevor ein Return geschieht. Wenn z.B. eine Prozedur A eine Prozedur B aufruft, die wiederum eine Prozedur C, so ist die Verschachtelungstiefe 3. Wenn allerdings B mit RETURN zu A zurückkehrt und dann C von A aufgerufen wird, beträgt die Verschachtelungstiefe nur 2.

Die vorgegebene Tiefe ist 50 Prozeduren. Dieser Wert kann aber erhöht werden, falls eine tiefere Verschachtelung denkbar ist. Eine rekursive Prozedur (die sich selbst aufruft) wird beispielsweise oft eine größere Tiefe benötigen. Die Stackgröße beträgt etwa 1KB für 10 Ebenen.

Der Vorgabewert von 50 Prozeduren verbraucht daher ca. 5KB an Verwaltungsaufwand und steigt proportional mit der Erhöhung von PROCEDURE DEPTH.

Da sowohl dieser Befehl als auch der STACK-Befehl die Größe des Stacks kontrollieren, sollte auch nur einer dieser Befehle verwandt werden (der letzte verwendete ist für BLINKER der maßgebliche). Programmierer, die eine andere Programmiersprache als CA-Clipper verwenden, sollten direkt das STACK-Kommando verwenden, da die Berechnung, die Blinker ausführt, um die echte Stackgröße mittels dieses Befehls zu ermitteln, an den Stack-Anforderungen von CA-Clipper-Prozeduren orientiert ist.

Blinker kann den Stack nur bei der Ausführung von Overlay-Code überprüfen. Doch kann es bei manchen Applikationen, die in den Root-Bereich gelinkte Drittanbieter-Bibliotheken benutzen, insbesondere in einer Netzwerkumgebung erforderlich sein, den Wert für PROCEDURE DEPTH zu erhöhen.Typische Symptome für eine zu geringe Verschachtelungstiefe sind plötzliches Sich-aufhängen einer Applikation, falsche Zeichen auf dem Bildschirm oder andere, nicht regelmäßig auftretende schwerwiegende Fehler.

Siehe auch: STACK

BLINKER SEGMENT THRESHOLD

Zweck: Verändert die standardmäßige Beschränkung der Segmentkomprimierung

Syntax: `BLINKER SEGMENT THRESHOLD <nuGrenze>`

Standard: <nuGrenze> ist 65500

Beispiel:
```
# Verändert die standardmäßige Beschränkung der
#  Segmentkomprimierung
BLINKER SEGMENT THRESHOLD 16384
```

Bemerkung: Beim Erzeugen einer Datei im neuen EXE-Format, d.h. bei DOS-Extender- oder Windows-Programmen, komprimiert der Blinker Code- und Datensegmente, um die Anzahl der Segmente in der abschließenden ausführbaren Datei zu verringern. Zur Laufzeit werden Daten und Code als komplette Segmente geladen und wieder friegegeben. Dies hat einen negativen Einfluß auf die Ausführungsgeschwindigkeit, wenn die Segmente häufig geswappt werden.

Mit Hilfe dieses Befehls kann die maximale Größe zusammengesetzter oder gepackter Segmente (in Bytes) angegeben werden, wodurch die Größe der zu swappenden Segmente reduziert werden kann.

Eine Verringerung der Segmentgröße ergibt natürlich eine größere Anzahl von Segmenten in der EXE-Datei.

Im neuen EXE-Format dürfen maximal 254 Segmente enthalten sein. Wenn die Segmentgröße zu niedrig angesetzt wurde, (und dadurch mehr als 254 Segmente benötigt werden), dann zeigt der Blinker zur Linkzeit eine Fehlermeldung 1141 an, die angibt, daß die Segmentbeschränkung für die neue EXE-Datei überschritten wurde.

CODECLASS

Zweck: Gibt an, daß zusätzliche Segmentklassen wie CODE behandelt werden sollen

Syntax: `CODECLASS <Klassenname> [,<Klassenname> ...]`

Standard: Kein

Beispiel:
```
# Behandelt die Segmente der Klassen MYCLASS1 und
#  MYCLASS2 als Code
CODECLASS MYCLASS1,MYCLASS2
```

Bemerkung: Über diesen Befehl können zusätzliche Segmentklassen angegeben werden, die vom Blinker als CODE behandelt werden sollen. Standardmäßig werden alle Klassen als CODE behandelt, deren Name auf CODE endet.

Hiermit kann abgesichert werden, daß Segmente in einer neuen .EXE-Datei als CODE gekennzeichnet werden.

Siehe auch: OVERLAY

CODEVIEW

Zweck: CodeView Debugger-Unterstützung

Syntax: CODEVIEW

Standard: Ausgeschaltet

Bemerkung: Der CODEVIEW-Befehl ermöglicht die globale und rückwirkende Verarbeitung von CodeView-Debug-Informationen unabhängig von der Position des CODEVIEW-Befehls in der Skriptdatei. Der CODEVIEW-Befehl entspricht einer CVON-Anweisung am Anfang der Skriptdatei.

Entwickler großer Overlay-Programme werden eine Beeinträchtigung der Performance ihres Overlay-Codes bemerken, wenn er nach Verwendung des CODEVIEW-Kommandos unter dem CodeView-Debugger läuft. Wenn die Geschwindigkeit zu diesem Zeitpunkt von Bedeutung ist, so sollten mittels CVON und CVOFF CodeView-Bereiche erzeugt werden, wo selektiv Debugging-Informationen für bestimmte Dateien, Module oder Bibliotheken einbezogen werden.

Die Standardversion von CodeView-Informationen, die in die .EXE-Datei geschrieben wird, ist 5, um eine höchstmögliche Kompatibilität mit anderen Debuggern zu gewährleisten. Ältere Versionen können mittels des CVVERSION-Befehls gesetzt werden (Weitere Details finden Sie in dem Abschnitt CVVERSION im Kapitel 9).

Hinweis: Der Einsatz dieses Befehls erfordert die Verwendung externer Overlays.

Siehe auch: CVOFF
CVON
CVVERSION
SECTION INTO

CVOFF

Zweck: Bearbeitung von CodeView-Informationen lokal abschalten

Syntax: `CVOFF`

Standard: Abgeschaltet

Bemerkung: Der CVOFF-Befehl beendet einen CodeView-Bereich und schaltet die Bearbeitung von CodeView-Informationen ab, bis ein neues CVON-Kommando innerhalb des Linkskripts auftaucht. CodeView-Bereiche werden verwendet, um die Menge des Overlay-Codes, für den CodeView-Debuggen eingeschaltet ist, zu begrenzen.

Große Overlay-Applikationen werden unter dem Debugger schneller ausgeführt, wenn die Menge des mit Debugging-Informationen versehenen Overlay-Codes durch CodeView-Bereiche begrenzt wird.

Siehe auch: CODEVIEW
CVON
CVVERSION

CVON

Zweck: Bearbeitung von CodeView-Informationen lokal einschalten

Syntax: `CVON`

Standard: Abgeschaltet

Beispiel:

```
FILE a,b,c
 SECTION INTO abc_xyz
 BEGINAREA
 FILE d,e
   #CodeView-Bereich für die Dateien f und g einrichten
   CVON
   FILE f,g
   #CodeView-Bereich beenden
   CVOFF
   LIB abc
   #CodeView-Bereich für xyz.lib einrichten
   CVON
   LIB xyz
   # CodeView-Bereich beenden
   CVOFF
 ENDAREA
 LIB funclib
```

Bemerkung: Der CVON-Befehl startet einen CodeView-Bereich und schaltet die Bearbeitung von CodeView Informationen ein, bis ein CVOFF-Kommando innerhalb des Linkskripts auftaucht. CodeView-Bereiche werden verwendet, um die Menge des Overlay-Codes, für den CodeView-Debuggen eingeschaltet ist, zu begrenzen.

Große Overlay-Applikationen werden unter dem Debugger schneller ausgeführt, wenn die zu debuggenden Bereiche durch CodeView-Befehle begrenzt werden.

Dateien, Module oder Bibliotheken innerhalb eines CodeView-Bereichs speichern alle ihre Symbol-Informationen zur Verwendung durch den Debugger im ausführbaren Programm ab. Dateien, Module oder Bibliotheken außerhalb eines CodeView-Bereichs legen nur ihre Public-Symbole und Modul-Grenzen im ausführbaren Programm ab.

Hinweis: Der Einsatz dieses Befehls erfordert die Verwendung externer Overlays

Siehe auch: CODEVIEW
CVOFF
CVVERSION
SECTION INTO

CVVERSION

Zweck: Stellt die Versionsnummer für die CodeView-Informationen ein

Syntax: `CVVERSION <0 | 1 | 2 | 5>`

Standard:

Bemerkung: CVVERSION stellt die Version der in die .EXE-Datei geschriebenen CodeView-Informationen ein. Die Versionsnummer der CodeView-Information bezieht sich nicht auf die Versionsnummer des CodeView-Debuggers, sondern gibt Aufschluß über die Version des CodeView-Informationsformats. Zum Zeitpunkt der Drucklegung gab es vier mögliche Versionen, 0, 1, 2 oder 5:

0 Alle Versionen des CodeView-Debuggers bis einschließlich Version 2.2

1 Die Version von CodeView, welche mit MS-Basic PDS 7.0 ausgeliefert wird.

2 Die Version von CodeView, die mit MS C 6.0 ausgeliefert wird.

5 Die Version von CodeView, die mit MS C/C++ 7.0, MS Visual C++ und MS Visual Basic für DOS 1.0 ausgeliefert wurde.

Blinker's Standardwert für diese Information ist 5, die neueste Version.

Sollten Sie Blinker anweisen, eine Versionsnummer zu verwenden, die niedriger liegt, als die Version, die Ihr CodeView-Debugger bearbeiten könnte, so kann es passieren, daß einige Möglichkeiten des Debuggers nicht mehr korrekt funktionieren. Verwenden Sie die höchste Versionsnummer, die zu Ihrem Debugger kompatibel ist.

Hinweis: Wenn Sie die CodeView-Informationen (Version 5) nutzen wollen, dann sollten Sie darauf achten, daß sich das Dienstprogramm CVPACK von Microsoft im Zugriff (Pfad) befindet.

Siehe auch: CODEVIEW

DEFBEGIN

Zweck: Bezeichnet den Beginn des Befehlsblockes für Moduldefinitionsbefehle

Syntax: `DEFBEGIN`

Standard: Kein

Beispiel:

```
# Gibt einen Programmnamen ohne .DEF-Datei an
DEFBEGIN
  NAME TEST
DEFEND
```

Bemerkung: Über diesen Befehl kann der Beginn eines Blocks mit Moduldefinitionsbefehlen kenntlich gemacht werden, die in einer Standard-Freeformat-Linkdatei enthalten sind. Dadurch kann man sich ersparen, zwei getrennte Linkdateien für DOS-Extender- oder Windows-Programme pflegen zu müssen. Alle nachfolgend eingetragene Linkdateizeilen werden bis zum zugehörigen DEFEND-Befehl als Moduldefinitionsdateibefehle behandelt.

Weitere Einzelheiten zu den gültigen Moduldefinitionsbefehlen finden Sie Kapitel 6, "Windows-Programme".

Siehe auch: DEFEND
DEFFILE

DEFEND

Zweck: Bezeichnet das Ende des Befehlsblockes für Moduldefinitionsbefehle

Syntax:
```
DEFEND
```

Standard: Kein

Beispiel:
```
# Gibt einen Programmnamen ohne .DEF-Datei an
DEFBEGIN
  NAME TEST
DEFEND
```

Bemerkung: Über diesen Befehl kann das Ende eines Blockes mit Moduldefinitionsbefehlen kenntlich gemacht werden, die in einer Standard-Freeformat-Linkdatei enthalten sind. Dadurch kann man sich ersparen, zwei getrennte Linkdateien für DOS-Extender- oder Windows-Programme pflegen zu müssen.

Weitere Einzelheiten zu den gültigen Moduldefinitionsbefehlen finden Sie Kapitel 6 "Windows-Programme".

Siehe auch: DEFBEGIN
DEFFILE

DEFFILE

Zweck: Gibt die Datei mit den Moduldefinitionsbefehlen an

Syntax:
```
DEFFILE <Dateiname>
```

Standard: Kein

Beispiel:
```
# Gibt die Moduldefinitionsdatei TEST an
DEFFILE TEST
```

Bemerkung: Dieser Befehl wird verwendet, um in einer Standard-Freeformat-Linkdatei den Namen einer Datei mit Moduldefinitionsbefehlen zu spezifizieren. Wurde keine Dateierweiterung angegeben, dann wird die Standarddateierweiterung .DEF verwendet.

Die Ausführung der Link-Skriptdatei wird unterbrochen, solange die angegebene Moduldefinitionsdatei (und alle darin verschachtelten Dateien) abgearbeitet wird. Danach wird die Ausführung auf der nächstfolgenden Zeile der Linkdatei fortgesetzt.

Weitere Einzelheiten zu den gültigen Moduldefinitionsbefehlen finden Sie in Kapitel 6 "Windows-Programme".

Siehe auch: DEFBEGIN
DEFEND

DEFINE

Zweck: Symbole spezifizieren, die zur Linkzeit auszuschließen sind

Syntax: `DEFINE <Symbol>[, <Symbol> ...]`

Beispiel: `DEFINE Alpha, Beta`

Bemerkung: Der DEFINE-Befehl wird verwendet, um die benannten Symbole auszuschließen und sie auf eine Dummy-Routine innerhalb des Blinker Overlay-Managers zu richten. Anstatt die Module, die diese Symbole beinhalten, in die resultierende .EXE-Datei aufzunehmen, läßt Blinker jedes DEFINE'te Symbol zu einer Routine des Overlay-Managers zeigen, die, sofern aufgerufen, den Laufzeitfehler 1213 anzeigt. Dieser Befehl kann (mit Vorsicht) verwendet werden , um das Einbinden von Routinen zu verhindern, die niemals aufgerufen werden.

Beachten Sie, daß, um eine Ausschließung eines Modules zu erreichen, alle seine Public-Symbole als DEFINE-Symbol angegeben sein müssen.

Gewährleisten Sie, daß ausschließlich Code-Symbole durch DEFINE ausgeschlossen werden. Sollten DEFINE'te Datenelemente beschrieben werden, kann der Overlay-Manager Schaden nehmen.

Beachten Sie bitte, daß die Verwendung von MIXCASE vor DEFINE zur Folge hat, daß alle Symbole auf Groß-Kleinschreibung hin unterschieden werden.

Siehe auch: MODULE
BLINKER EXECUTABLE NODELETE
DOS-Real Mode-Laufzeitfehler (12xx)

DEFLIB

Zweck: Verwenden der vorgegebenen Bibliothek-Sucheinträge

Syntax: `DEFLIB`

Standard: Eingeschaltet

Bemerkung: Der DEFLIB-Befehl sorgt dafür, daß Blinker Bibliotheken lokalisiert und einbindet, die in den Objekt-Modulen angegeben sind. Einige Compiler fügen Informationen über Librarynamen, die der Linker benötigen könnte, in jede Objektdatei ein. Dieser Befehl ist standardmäßig eingeschaltet.

Siehe auch: NODEFLIB

DOSSEG

Zweck: Verwendet die Microsoft DOS-Segmentanordnung

Syntax: `DOSSEG`

Standard: Ausgeschaltet

Bemerkung: Über den Befehl DOSSEG wird angegeben, daß der Blinker die Programm-Segmente in Übereinstimmung mit dem Segmentanordnungsschema der Hochsprachen-Compiler von Microsoft anordnen soll. Die meisten Compiler, für die diese Segmentanordnung benötigt wird, enthalten in jeder erzeugten .OBJ-Datei oder in der Sprachbibliothek eine DOSSEG-Anweisung, so daß dieser Befehl unter normalen Umständen nicht erforderlich ist.

Für WATCOM C/C++ wird ein expliziter DOSSEG-Befehl für alle Programme benötigt.

ECHO

Zweck: Skript-Befehle oder Text auf dem Bildschirm ausgeben

Syntax: `ECHO [ON | OFF |] [<cString>]`

Standard: OFF

Bemerkung: ECHO <cString> gibt den String <cString> auf dem Bildschirm aus, unabhängig vom Status von ECHO und ohne diesen zu verändern. ECHO ON <cString> schaltet die Bildschirmausgabe ein und bringt zusätzlich den String <cString> auf dem Bildschirm aus. ECHO OFF schaltet die Ausgabe von Bildschirmmeldungen ab und der Rest der Zeile wird ignoriert.

ENDAREA

Zweck: Gibt das Ende eines Overlay-Bereichs an

Syntax: `ENDAREA`

Bemerkung: Der Befehl ENDAREA gibt dem Linker das Ende eines Overlay-Bereiches an, wie es im Abschnitt zu BEGINAREA beschrieben worden ist.

Siehe auch: BEGINAREA
SECTION INTO

EXTDICTIONARY

Zweck: Verwenden von erweiterten Verzeichnissen in den Bibliotheken einschalten, sofern welche vorhanden sind.

Syntax: `EXTDICTIONARY`

Standard: Eingeschaltet

Bemerkung: Dieser Befehl sorgt dafür, daß Blinker das erweiterte Verzeichnis am Ende einer Bibliothek verwendet, welches durch einige Bibliothek-Programme an die Bibliothek angehängt wird. Dieses erweiterte Verzeichnis enthält Kreuzverweise des Bibliothek-Inhaltes und wird von Blinker verwendet, um die Bibliothek-Bearbeitung zu beschleunigen. Nicht alle Bibliothek-Programme stellen ein erweitertes Verzeichnis zur Verfügung. In diesen Fällen bearbeitet Blinker die Bibliotheken etwas langsamer.

Siehe auch: NOEXTDICTIONARY

EXTRAMEM

Zweck: Beschränkt die Größe des Speichers, der durch ein Real Mode Programm verwendet werden darf

Syntax: `EXTRAMEM <nuGröße>`

Standard: NuGröße ist 1 MB - (physische Abbildgröße)

Bemerkung: Über den Befehl EXTRAMEM wird die Größe des zusätzlichen Speichers beschränkt, den DOS für ein Programm belegt. Der Parameter enthält die Größenangabe (als Dezimalzahl der Paragraphs jenseits der Abbildgröße) des größten Speicherblockes, in den DOS das Programm laden soll. Dieser Befehl entspricht der Option /CPARM von MSLink.

Unter anderem wird dieser Befehl dazu verwendet, um die Größe des Near Heap in Medium- und Large Modell-Programmen zu verringern.

Es gibt drei Eckwerte, die mit den Speicheranforderungen einer ausführbaren DOS-Datei verbunden sind:

a) Die physische Größe des Hauptteils einer ausführbaren Datei.

b) Die zusätzliche Menge an Speicher, die für nichtinitialisierte Daten benötigt werden.

c) Zusätzlicher Speicherbedarf des Programms. Dieser Wert beträgt normalerweise 1 MB. Dadurch wird erreicht, daß DOS während des Ladens den größten verfügbaren zusammenhängenden Speicherblock für das Programm belegt.

Der Wert (a+b) wird als Ladegröße bezeichnet. Diese entspricht der Speichermenge, die DOS zum Laden des Programms in den Speicher benötigt.

Der Befehl EXTRAMEM spezifiziert eine Anzahl von Abschnitten, die für das Programm ZUSÄTZLICH zu der physischen Abbildgröße belegt werden. Dieser Wert muß mindestens so groß wie (b) sein. Ist die Anzahl der angegebenen Abschnitte KLEINER als (b) ist, dann vergrößert der Blinker das EXTRAMEM-Abbild automatisch auf den Minimalwert.

Siehe auch: MEMORY

FILE

Zweck: Angabe der Objektdatei(en)

Syntax: `FILE [L:][Pfad]<Dateiname> [, [L:][Pfad]<Dateiname> ...]`

Beispiel: `FILE Myapp1, D:\APPS\OBJ\Myapp2`

Bemerkung: Der Befehl FILE gibt Blinker die Namen einer oder mehrer .OBJ-Dateien an, die an der spezifizierten Stelle in die Ausgabedatei eingebunden werden sollen. Die vorgegebene Erweiterung .OBJ muß nicht mit angegeben werden. Die erste aufgeführte Datei ist normalerweise das Hauptmodul des Programms.

Falls mehrere Objektdateien hintereinander angegeben werden, müssen sie jeweils durch ein Komma voneinander getrennt sein. Blinker interpretiert eine Wagenrücklauf-/ Zeilenvorschubkombination als Ende des FILE-Befehls, deshalb muß der FILE-Befehl am Anfang jeder Zeile neu angegeben werden.

Alle in FILE-Befehlen aufgelisteten .OBJ-Dateien werden in die Applikation mit eingebunden, ob sie benötigt werden oder nicht. Um nur die tatsächlich benötigten .OBJ-Dateien in die Applikation einzuschließen, sollten Sie eine Bibliothek-Datei (.LIB) mit Hilfe eines Bibliothek-Managers (z.B. MS LIB.EXE) erzeugen. Diese Bibliothek kann nun mit dem Befehl LIBRARY in die Applikation eingebunden werden, wobei dann nur die wirklich benötigten Objektdateien aus der .LIB-Datei übernommen und gelinkt werden.

Blinker sucht jede .OBJ-Datei zuerst im aktuellen Verzeichnis und dann in der angegebenen Reihenfolge in jedem Pfad, der in der Umgebungsvariablen OBJ definiert ist. Alternativ dazu kann jede .OBJ-Datei mit komplettem Pfad aufgeführt werden.

Für eine Bibliotheksdatei kann der Befehl FILE dazu verwendet werden, um zu erzwingen, daß alle in der Bibliothek enthaltenen Module in die .EXE-Datei übernommen werden. In diesem Fall sind die Angabe der Dateierweiterung .LIB sowie eventuell eine vollständige Pfadangabe erforderlich, z.B. FILE pfad\CLD.LIB.

Siehe auch: LIBRARY (Linkskriptbefehl)

LIBRARY (Linkskriptbefehl)

Zweck: Angabe eines oder mehrerer Namen von Programm-Bibliotheken

Syntax: `LIBRARY [L:][Pfad]<Bibliotheksname> [, [L:][Pfad]<Bibliotheksname> ...]`

Standard: Durch den Compiler vorgegebene Standard-Bibliotheken

Beispiel: `LIB C:\LIBS\Mylib1, Mylib2, D:\OTHER\Mylib3`

Bemerkung: Mit dem Befehl LIBRARY werden dem Linker die Namen einer oder mehrerer Programm-Bibliothek-Dateien angegeben, die zum korrekten Linken der Applikation benötigt werden. Die vorgegebene Erweiterung .LIB muß nicht mit angegeben werden. Falls mehrere

Bibliothek-Dateien hintereinander angegeben werden, müssen sie jeweils durch ein Komma voneinander getrennt sein.

Bibliotheken werden insofern anders als FILE-Befehle verarbeitet, als lediglich die von der Applikation benötigten Module aus einer Bibliothek eingebunden werden, d.h. ein Modul einer Bibliothek wird dann miteingebunden, wenn eine Public-Definition in diesem Bibliothek-Modul eine externe Referenz auflöst. Im FILE-Befehl aufgeführte Objektdateien werden auch dann miteingebunden, wenn sie nicht benötigt werden.

Der LIBRARY-Befehl kann sowohl für ins Root als auch für ins Overlay zu ladende Bibliotheken verwendet werden. Soll die Bibliothek ins Overlay gelegt werden, so muß sie innerhalb einer BEGINAREA / ENDAREA-Sequenz angegeben werden.

Siehe auch: DEFLIB
NODEFLIB
SEARCH
FILE
MODULE

MAP

Zweck: Ausgabe einer Segment-Übersicht des ausführbaren Programmes

Syntax: `MAP [=[L:][Pfad]<Dateiname>] [S][,A][,X]`

Standard: Ausgeschaltet

Beispiel:
```
# erstellen einer Übersicht aller Public-Symbole
MAP=C:\TEMP\Mymap A
```

Bemerkung: Der MAP-Befehl gibt dem Linker an, daß er für die zu erzeugende .EXE-Datei eine Map-Datei erstellen soll. Die Map-Datei erhält standardmäßig den gleichen Namen wie die .EXE-Datei, allerdings mit der Erweiterung .MAP. Wenn Sie einen alternativen Dateinamen verwenden möchten, geben Sie ihn im MAP-Befehl an. Die Map-Datei kann unterschiedliche Auswertungsstufen enthalten.

Die drei möglichen Befehlsoptionen können in belieber Reihenfolge angegeben werden und haben folgende Bedeutung:

S - alle Segmente - Listet Adresse, Länge und Namen jedes Segmentes der ausführbaren Datei auf. Dies ist die Standard-Option, wenn keine angegeben werden.

A - Public-Symbole nach Adressen geordnet. Listet die Public-Symbole und ihre Speicheradressen auf.

Die A-Option kann verwendet werden, um eine Map-Datei zu erzeugen, die für das Debuggen von C- und Assembler-Programmen unter Benutzung von Debuggern, welche nicht die CodeView-Information unterstützen, benötigt wird.

X - Segmentdetails je Modul - listet Adresse, Länge, Name und Attribute jedes Segments von jedem in das Programm gelinkte Modul auf.

Sehen Sie bitte auch im Anhang, um weitere Informationen zu den MAP Ausgaben verschiedener Programmtypen zu bekommen.

Siehe auch: BLINKER MESSAGE DUPLICATES
NWIDTH
VERBOSE
Anhänge

MEMORY

Zweck: Beschränkt die Größe des Speichers, der für ein Real Mode Programm belegt wird

Syntax: `MEMORY <nuLimit>`

Standard: <nuLimit> ist 1 MB

Bemerkung: Über den Befehl MEMORY wird die Größe des Speichers beschränkt, den DOS für ein Programm belegt. Der Parameter enthält die Größenangabe (als Dezimalzahl der Abschnitte) des größten Speicherblockes, in den DOS das Programm laden soll.

Unter anderem wird dieser Befehl dazu verwendet, um die Größe des Near Heap in Medium- und Large Modell-Programmen zu verringern.

Es gibt drei Eckwerte, die mit dem Speicherbedarf einer ausführbaren DOS-Datei verbunden sind:

a) Die physische Größe des Rootteils einer ausführbaren Datei.

b) Die zusätzliche Menge an Speicher, die für nichtinitialisierte Daten benötigt werden.

c) Zusätzlicher Speicherbedarf des Programms. Dieser Wert beträgt normalerweise 1 MB. Dadurch wird verursacht, daß DOS während des Ladens den größten verfügbaren zusammenhängenden Speicherblock für das Programm belegt.

Der Wert (a+b) wird als Ladegröße bezeichnet. Diese entspricht der Speichermenge, die DOS zum Laden des Programms in den Speicher benötigt.

Der Befehl MEMORY gibt die Gesamtanzahl der Abschnitte an, die für ein Programm belegt werden sollen (a+b+c). Diese muß mindestens so groß sein wie (a+b). Ist der angegebene Wert KLEINER als (a+b), dann vergrößert der Blinker das MEMORY-Abbild automatisch auf den Minimalwert.

Siehe auch: EXTRAMEM

MIXCASE

Zweck: Unterscheidung zwischen Groß-und Kleinschreibung

Syntax: `MIXCASE [EXPORT | Import]`

Standard: Ausgeschaltet

Bemerkung: Der MIXCASE-Befehl erzwingt, daß Blinker alle Public- und External-Symbole während des Linkens nach Groß-und Kleinschreibung unterscheidet, anstatt diese in Großbuchstaben (UPPERCASE) umzuwandeln. Dieses Kommando ist von speziellem Interesse für Programmierer, die auch C einsetzen, wo diese Unterscheidung von Bedeutung sein kann.

Hinweise für DOS Extender Programme:

MS Link konvertiert Exporte von 16 Bit Windows-Programmen immer zu Großbuchstaben. Daher müssen Sie darauf achten, daß auch alle Importe in Großbuchstaben sind, wenn Sie von einer 16 Bit Windows DLL, die mit MS Link zu einem DOS Extender Programm gelinkt ist, importieren wollen.

Hinweise für 16 Bit Windows-Programme:

Standardmäßig werden alle Namen einer Exportdefinition in Moduldefinitionsdateien für 16 Bit Windows-Programme zu Großbuchstaben konvertiert, wenn sie verarbeitet werden. Wenn MIXCASE EXPORT im Linkskript angegeben wurde, dann wird die Groß-/Kleinschreibung bei Exportnamen unterschieden. Verwenden Sie diese Option mit Vorsicht, da sie zu unaufgelösten Symbolen während der Link- oder Laufzeit führen kann.

Standardmäßig werden alle Namen einer Importdefinition in Moduldefinitionsdateien für 16 Bit Windows-Programme zu Großbuchstaben konvertiert, wenn sie verarbeitet werden. Wenn MIXCASE IMPORT im Linkskript angegeben wurde, dann wird die Groß-/Kleinschreibung bei Importnamen unterschieden. Verwenden Sie diese Option mit Vorsicht, da sie zu unaufgelösten Symbolen während der Link- oder Laufzeit führen kann.

Diese Optionen werden erst ab der Stelle wirksam, an der sie im Linkskript stehen. Daher sollten sie normalerweise nahe am Anfang eines Linkskriptes stehen, um sicherzustellen, daß alle Symbole korrekt konvertiert werden.

Siehe auch: UPPERCASE

MODULE

Zweck: Positionierung einzelner Bibliothek-Module

Syntax:
```
MODULE <Modul> [,<Modul> ...] [FROM <Bibliotheksname>]
```

Beispiel:
```
# Modul COLOR an die aktuelle Position bewegen
MODULE COLOR
```

Bemerkung: Der MODULE-Befehl erlaubt es, einzelne Module entweder ins Root oder ins Overlay zu verfrachten, unabhängig davon, ob ihre Bibliotheken oder Objektmodule innerhalb eines Overlay-Bereichs angegeben werden oder nicht.

Der Name des Moduls verweist auf ein Objektmodul innerhalb der Bibliothek, die innerhalb der Skriptdatei über einen LIB- oder SEARCH-Befehl angegeben wurde. Über die Ausgaben des Blinker-Befehls VERBOSE werden alle Module aufgeführt, die während des Linkens bearbeitet werden.

Die Position des MODULE-Kommandos innerhalb des Link-Skripts bestimmt die Position des benannten Moduls im ausführbaren Programm. Ist das MODULE-Kommando im Root, so wird das angegebene Modul entsprechend im Root plaziert; ist es innerhalb eines Overlay-Bereichs, so wird das Modul entsprechend im Overlay plaziert. Es können mehrere Module auf derselben Zeile angegeben werden, sofern sie durch ein Komma getrennt werden.

Die Bibliothek- und Modulnamen sollten ohne Pfadangaben angegeben werden, anderenfalls werden die Pfadangaben ignoriert. Das VERBOSE-Kommando gibt Aufschluß darüber, wie der korrekte Modulname lautet, da dieser während des Linkens angezeigt wird. Der gleiche Name wird auch von Microsoft's LIB.EXE verwendet.

MODULE-Befehle sind unabhängig von Groß-Kleinschreibung (große und kleine Buchstaben werden gleich behandelt), solange das MIXCASE-Kommando nicht vor einer MODULE-Anweisung verwendet wurde.

Das MODULE-Kommando hat einen zweiten Verarbeitungsmodus, wobei eine Quell-Bibliothek durch den Zusatz FROM angegeben wird. Dies ermöglicht es, Modulen spezifizierter Bibliotheken den Vorzug gegenüber anderen zu geben, und gleichzeitig die normale MODULE-Funktion auszuführen - die Plazierung von Routinen im Root- oder Overlay-Bereich zu verändern. Dies kann Konflikte zwischen zwei Bibliotheken lösen, die gleichnamige Module enthalten.

Es ist ratsam, die Option FROM nur wenn nötig zu verwenden, da sie die normale Linksequenz modifiziert.

```
MODULE DBEDIT,MEMOEDIT FROM EXTEND
```

Dieser Befehl erzwingt, daß die Module DBEDIT und MEMOEDIT aus der EXTEND.LIB gleichnamigen Modulen, die eventuell vorher schon in anderen Bibliotheken oder Dateien gefunden wurden, vorgezogen und zu der aktuellen Stelle der Skriptdatei bewegt werden.

```
MODULE MEMOEDIT FROM NUL
```

Dieser Befehl sorgt dafür, daß das Modul MEMOEDIT nicht miteingebunden wird, da Blinker nach einer Bibliothek NUL.LIB sucht, die nicht existiert. Nimmt jedoch ein anderes Modul explizit Bezug auf MEMOEDIT, so werden Sie ein 'nicht aufgelöstes External' erhalten. Um dies zu umgehen, verwenden Sie das DEFINE-Kommando anstelle des

MODULE-Kommandos und definieren Sie die nicht aufgelösten Public-Symbole innerhalb des Moduls.

Siehe auch: DEFINE
BLINKER MESSAGE DUPLICATES
LIBRARY (Link-skriptbefehl)
VERBOSE

MURPHY

Zweck: Erzeugen einer ‘Worst Case’-Overlay-Situation

Syntax: `MURPHY`

Standard: Ausgeschaltet

Bemerkung: Der Befehl MURPHY sorgt dafür, daß der Overlay-Manager eine Situation zu schaffen versucht, die jeden verborgenen Overlay-Fehler (Programmierfehler) während des Testens oder Debuggens aufzeigt.

Wenn MURPHY eingeschaltet ist, läßt der Overlay-Manager nur jeweils ein einziges Overlay im Speicher zu. Dies ermöglicht es dem Programmierer, sich ‘daneben benehmende’ Abhängigkeiten zwischen Overlays herauszufinden und zu korrigieren.

Der Overlay-Manager füllt auch allen ungenutzten Raum im Overlay-Pool mit INT-3 Instruktionen, mit einem 'far jump' zu einer Fehlerausgabe-Routine innerhalb des Overlay-Managers, die ans Ende des Overlay-Bereiches gehängt wird.

Wird ein Programm unter einem Debugger ausgeführt und ein falscher Sprung zu einer ungültigen Adresse innerhalb des Overlay-Bereichs durchgeführt, wird die INT 3-Anweisung dafür sorgen, daß der Debugger an dieser Adresse stoppt.

Läuft das Programm nicht unter einem Debugger, so werden alle INT 3-Instruktionen und der 'far jump' zu der Fehlerroutine im Overlay-Manager ausgeführt. An diesem Punkt gibt der Overlay-Manager eine Meldung aus und beendet das Programm.

Das MURPHY-Kommando kann die Ablaufgeschwindig-keit ihrer Applikation stark beeinträchtigen.

NOBELL

Zweck: Den Piepser am Ende des Linkvorgangs unterdrücken

Syntax: `NOBELL`

Standard: Ausgeschaltet

Bemerkung: Der Befehl NOBELL sagt dem Linker, daß nach Vollendung des Link-Prozesses kein Signalton ausgegeben werden soll.

NODEFLIB

Zweck: Ignoriert vorgegebene Bibliothek-Sucheinträge

Syntax: `NODEFLIB`

Standard: Ausgeschaltet

Bemerkung: Das NODEFLIB-Kommando gibt Blinker an, daß die in Objektdateien gefundenen vorgegebenen Bibliothek-Sucheinträge nicht verwendet werden sollen. Einige Compiler fügen Informationen über die Namen der Bibliotheken, die der Linker später benötigen könnte, in jede erstellte Objektdatei ein. Blinker's Standardeinstellung sorgt dafür, daß nach solchen Einträgen gesucht wird, solange nicht NODEFLIB verwendet wird.

Siehe auch: DEFLIB

NOEXTDICTIONARY

Zweck: Verwendung erweiterter Verzeichnisse in den Bibliotheken ausschalten

Syntax: `NOEXTDICTIONARY`

Standard: Ausgeschaltet

Bemerkung: Dieser Befehl bewirkt, daß Blinker das durch einige Bibliothek-Programme an Bibliotheken angehängte erweiterte Verzeichnis nicht verwenden soll. Dieses erweiterte Verzeichnis enthält Querverweise zum Bibliothek-Inhalt und wird von Blinker verwendet, um die Bearbeitung von Bibliotheken zu beschleunigen.

Wird dieser Befehl verwendet, bearbeitet Blinker die Bibliotheken etwas langsamer. Es gibt einige Bibliothek-Manager, die dafür bekannt sind, ungültige Verzeichnisse zu erstellen, was zu irreführenden Resultaten während der Linkzeit führen kann. Dieses Kommando sorgt dafür, daß solche Bibliotheken korrekt bearbeitet werden können.

Der Befehl hat globale Gültigkeit für alle während des Linklaufs bearbeiteten Bibliotheken.

Siehe auch: EXTDICTIONARY

NOPACKCODE

Zweck: Deaktivieren des Packen von NEW EXE Code-Segmenten

Syntax: `NOPACKCODE`

Standard: Ausgeschaltet

Bemerkung: Dieser Befehl gibt an, daß Blinker bei der Erzeugung von Dateien im NEW EXE Format (Windows- und DOS Extended-Programme) Code-Segmente nicht packen soll.

Normalerweise packt Blinker die Code-Segmente eines Programmes im NEW EXE Format,

da dieses Format nur eine begrenzte Anzahl von Segmenten (254) erlaubt.

Siehe auch: PACKCODE

NOPACKDATA

Zweck: Deaktivieren des Packen von NEW EXE Data-Segmenten

Syntax: `NOPACKDATA`

Standard: Eingeschaltet

Bemerkung: Dieser Befehl gibt an, daß Blinker bei der Erzeugung von Dateien im NEW EXE Format (Windows- und DOS Extended-Programme) Data-Segmente nicht packen soll.

Standardmäßig packt Blinker die Data-Segmente eines Programmes im NEW EXE Format nicht.

Siehe auch: PACKDATA

NOTABLEOFCONTENTS

Zweck: Inhaltsverzeichnisse der Bibliotheken bleiben unberücksichtigt

Syntax: `NOTABLEOFCONTENTS`

Standard: Ausgeschaltet

Bemerkung: Dieser Befehl zwingt Blinker, alle Bibliotheken zu durchsuchen (wie wenn sie mit SEARCH angegeben wären), d.h. keine Inhaltsverzeichnisse der Bibliotheken zu berücksichtigen.
Unter bestimmten ungewöhnlichen Umständen kann dieser Befehl verwendet werden, um die Reihenfolge zu emulieren, in der Borlands TLINK Linker die Module innerhalb einer Bibliothek verarbeitet.

Dieser Befehl wirkt global für die Verarbeitung aller Bibliotheken.

NWIDTH

Zweck: Angabe der Spaltenbreite von Symbol- und Segmentnamen in der .MAP-Datei

Syntax: `NWIDTH <nuBreite>`

Standard: 23

Bemerkung: Dieser Befehl bestimmt die maximale Breite der Spalten für Symbol- und Segmentnamen in der .MAP-Datei. Symbolnamen, die länger sind als in NWIDTH angegeben, werden trozdem mit voller Länge und einer nachfolgenden Leerstelle angezeigt, aber nachfolgende Spalten in derselben Zeile sind dann nicht mehr ausgerichtet.

Siehe auch: MAP

OUTPUT

Zweck: Gibt den Namen der ausführbaren Ausgabedatei an

Syntax: `OUTPUT [L:][Pfad]<Dateiname>`

Standard: <Erstes Objekt>.EXE

Beispiel:
```
# erstelle eine gelinkte Datei namens OUT.EXE
OUTPUT OUT.EXE
```

Bemerkung: Der OUTPUT-Befehl gibt dem Linker den Namen der zu erzeugenden ausführbaren Ausgabedatei (mit Laufwerk und Pfad) an. Falls OUTPUT nicht angegeben wurde, benutzt Blinker den Namen und Pfad der nach dem ersten FILE-Befehl in einer Skriptdatei (oder in der Befehlszeile) aufgeführten Objektdatei.

Die OUTPUT-Datei muß keine .EXE-Erweiterung haben. Falls keine Erweiterung angegeben ist, wird automatisch die Dateinamenserweiterung .EXE angehängt.

Siehe auch: READONLY

OVERLAY

Zweck: Gibt an, welche Segmentklassen ins Overlay gelegt werden sollen

Syntax: `OVERLAY <Klassenname> [,<Klassenname> ...]`

Standard: Die Klasse des ersten Segmentes, welches während des Linkens gefunden wird.

Beispiel: `OVERLAY CODE`

Bemerkung: Mit dem Befehl OVERLAY wird dem Linker angegeben, welche Segmentklasse(n) der Objektmodule in den Overlay-Sektionen abgelegt werden soll(en).

Die Standard-Overlay-Klasse ergibt sich aus der ersten während des Linkens bearbeiteten Segmentklasse und wird, sofern VERBOSE verwendet wurde, nach Ende des ersten Durchgangs auf dem Bildschirm angezeigt. Bei Hochsprachen-Compilern sollte sich der Gebrauch dieses Befehls erübrigen, da sie automatisch alle Code-Segmente an den Anfang der Objektdateien stellen.

Assembler-Programmierer haben die vollständige Kontrolle über Segmentklassen und deren Reihenfolge in den Objektdateien. Daher kann dieser Befehl für sie von Nutzen sein, um Code, der sich nicht in Segmenten der Klasse CODE befindet, ins Overlay zu legen.

Blinker unterstützt nur dynamisches Overlaying von Code, daher benutzen Sie dieses Kommando bitte mit Vorsicht und versuchen Sie nicht, Datensegmente in ein Overlay zu legen.

Siehe auch: CODECLASS

PACKCODE

Zweck: Aktivieren des Packen von NEW EXE Code-Segmenten

Syntax: `PACKCODE [nuSchwelle]`

Standard: <nuSchwelle> ist 65500 Byte

Beispiel:
```
# Code-Segmente packen bis zu maximal 32 KB je Segment
PACKCODE 32768
```

Bemerkung: Dieser Befehl veranlaßt Blinker, Code-Segmente zu kombinieren, wenn Dateien im NEW EXE-Format (DOS Extended- oder Windows-Programme) erzeugt werden. Der optionale Parameter ist die maximale Größe für ein Segment, bevor ein neues begonnen wird. Das Packen von Code ist voreingestellt auf eine Grenze von 65500 Bytes. Dies ist auch der Voreinstellwert, wenn kein Parameter angegeben wird.

Siehe auch: NOPACKCODE

PACKDATA

Zweck: Aktivieren des Packen von NEW EXE Data-Segmenten

Syntax: `PACKDATA [nuGrenzwert]`

Standard: Ausgeschaltet

Beispiel:
```
# Data-Segmente packen bis zu maximal 32 KB je Segment
PACKDATA 32768
```

Bemerkung: Dieser Befehl veranlaßt Blinker, Data-Segmente zu kombinieren, wenn Dateien im NEW EXE-Format (DOS Extended- oder Windows-Programme) erzeugt werden. Der optionale Parameter ist die maximale Größe für ein Segment, bevor ein neues begonnen wird. Das Packen von Data ist in der Voreinstellung deaktiviert, außer wenn die Symboltabellen-Komprimierung für CA-Clipper 5.x-Programme aktiviert ist.

Der Standardwert für <nuGrenzwert> ist, wenn nichts anderes angegeben ist, 65535 Bytes.

Siehe auch: NOPACKDATA

RC (Linkskriptbefehl)

Zweck: Bestimmt eine Resource-Datei, die in die .EXE-Datei eingefügt wird.

Syntax:
```
RC [-m] resdateiname[.RES]
```

Beispiel:
```
# Die Resourcedatei MYRES.RES in die Ausgabedatei einfügen
RC MYRES
```

Bemerkung: Dieser Befehl gibt an, daß die Resource-Datei resdateiname in die .EXE-Datei eingefügt wird.

Durch die Option -m wird Blinker dazu veranlaßt, das Mehrinstanzenflag zu setzen. Andere Optionen, die üblicherweise mit dem Resource-Compiler gesetzt werden, können durch die Verwendung von Moduldefinitionsbefehlen gesetzt werden.

Der Resource-Compiler WRC von WATCOM benötigt die Option -zm, um MS-DOS 16-Bit-Ressourcen auszugeben, die mit dem Blinker-Befehl RC verwendet werden können. WRC muß für den zweiten Resource-Compilerlauf mit 32-Bit-Programmen verwendet werden, da er ein nicht standardisiertes 32-Bit-Resource-Dateiformat verwendet.

READONLY

Zweck: Ausführbares Programm mit dem Nur-Lesen-Status versehen

Syntax:
```
READONLY
```

Standard: Ausgeschaltet

Bemerkung: Das READONLY-Kommando veranlaßt Blinker, eine Ausgabedatei zu erstellen, die mit dem ‘Nur-Lesen’-Dateiattribut markiert ist. Dieser Befehl ist für die Entwickler interessant, die ihr Programm innerhalb einer Netzwerkumgebung linken und testen, da er ‘sharing-violation’-Fehler bei der Ausführung des Programms von mehreren Stationen aus verhindert.

Beachten Sie, daß das Attribut beim Kopieren der Datei verlorengeht.

Siehe auch: OUTPUT

SEARCH

Zweck: Schaltet den mehrfachen Suchdurchlauf nach ‘unaufgelösten Symbolen’ ein

Syntax:
```
SEARCH [L:][Pfad]<Bibliotheksname> [, [L:][Pfad]<Bibliotheksname> ...]
```

Standard: Ausgeschaltet

Beispiel:
```
SEARCH C:\LIBS\MYLIB1, D:\ABC\LIB2
```

Bemerkung: Der SEARCH-Befehl ist eine alternative Form des LIBRARY-Befehls, arbeitet aber etwas

anders.

Während eines normalen Linklaufes durchkämmt Blinker eine Bibliothek mehrfach, bis er keine weiteren externen Referenzen mehr auflösen kann, worauf er mit der Bearbeitung der nächsten Bibliothek beginnt. Wenn nach der Bearbeitung aller Bibliotheken noch unaufgelöste externe Referenzen existieren, wird Blinker die Bibliotheken einem weiteren Durchgang unterziehen usw..

Wenn Blinker auf ein SEARCH-Kommando trifft, lädt er alle Symbolnamen der Bibliothek in eine Tabelle im Speicher. Dies hat den Effekt, daß alle bisherigen unaufgelösten und alle zukünftigen externen Refernzen eher von der ge-SEARCH-ten (falls sie dort definiert sind) als von einer nachfolgenden Bibliothek aufgelöst werden.

Die Verwendung des SEARCH-Befehls sollte auf die Fälle beschränkt bleiben, wo es unbedingt nötig ist, da der Linkprozeß durch diesen Befehl verlangsamt wird und große Mengen Speicher während des Linkens benötigt werden.

Siehe auch: LIBRARY (Linkskriptbefehl)

SECTION INTO

Zweck: Gibt Dateien für ein externes Overlay an

Syntax:
```
SECTION INTO <Overlayname> [FILE [L:][Pfad]<Dateiname>[, [L:]
 [Pfad]<Dateiname>...] ]
```

Standard: Ausgeschaltet

Beispiel:
```
FILE MAIN
 BEGINAREA
 SECTION INTO OVL1
   FILE ABC, DEF, GHI      # in OVL1
 SECTION INTO OVL2
   FILE JKL, MNO, PQR      # in OVL2
 ENDAREA
```

Bemerkung: Der Befehl SECTION INTO gibt den Namen einer externen Overlay-Datei zur Aufnahme der Overlays an. Diese Datei kann jede beliebige Erweiterung außer .EXE haben. Ist keine Erweiterung angegeben, so wird automatisch .OVL angenommen. Der erste SECTION INTO-Befehl muß dem ersten FILE-Befehl im Overlay vorausgehen, sonst wird ein internes Overlay innerhalb der OUTPUT-Datei erzeugt und alle SECTION INTO-Befehle werden ignoriert. Beachten Sie, daß mit diesem Befehl noch keine Overlays erzeugt werden, sondern er bewirkt nur , daß Overlays (soweit vorhanden) außerhalb der .EXE-Datei gespeichert werden (in .OVL Dateien). Die Befehle BEGINAREA und ENDAREA erzeugen Overlays.

Wenn eine Overlay-Datei spezifiziert worden ist, wird jeder Overlay-Code in dieser Datei abgelegt, bis ein SECTION INTO-Befehl mit einem anderen Namen entdeckt wird. Dann wird eine neue Overlay-Datei erzeugt. Die Ausgabe kann durch Verwendung mehrerer SECTION INTO-Befehle auf die Overlay-Dateien verteilt werden.

Sobald ein internes Overlay erzeugt worden ist, werden alle SECTION INTO-Befehle ignoriert und alle Overlay-Prozeduren intern gespeichert. Wenn ein externes Overlay

erzeugt worden ist, kann kein internes Overlay mehr angelegt werden. Blinker kann in der vorliegenden Version maximal 15 externe Overlays anlegen.

Während der Laufzeit der Applikation wird jeder Overlay-Datei ein File Handle zugeordnet. Es kann daher notwendig sein, die Anzahl der verfügbaren File Handles zu erhöhen.

Siehe auch: BEGINAREA
ENDAREA

STACK

Zweck: Gibt die Stackgröße der .EXE-Datei an

Syntax: `STACK <nuGröße>`

Standard: Durch den Compiler festgelegt

Beispiel:
```
STACK 5120    # Stackgröße auf 5120 Bytes setzen
```

Bemerkung: Der Befehl STACK gibt dem Linker die von der .EXE-Datei benötigte Größe für den Stack als Dezimalzahl an.

Beachten Sie bitte, daß die Größe bei BLINKER als Dezimalzahl angegeben wird, während sie bei Microsoft Link als Hexadezimalzahl definiert wird. Eine Stackgröße von 1000 bei Microsoft Link würde daher einer Stackgröße von 4096 bei BLINKER entsprechen.

Anmerkung für Borland C++:

Die Stackgröße eines Borland C++ Programmes muß innerhalb des Programmcodes festgelegt werden, nicht durch einen STACK Befehl des Linkers. Schauen Sie bitte in die Borland Dokumentation für weitere Einzelheiten.

Anmerkung für CA-Clipper:

CA-Clipper-Entwickler sollten die Stackgröße erhöhen, indem sie mittels des Befehls BLINKER PROCEDURE DEPTH die maximale Verschachtelungstiefe der Applikation einstellen.

BLINKER's Standard-Stackgröße beträgt beim Linken von CA-Clipper-Programmen 5120 bytes, was einer PROCEDURE DEPTH von etwa 50 entspricht.

Siehe auch: BLINKER PROCEDURE DEPTH

UPPERCASE

Zweck: Wandelt alle Symbole in Großbuchstaben um

Syntax: `UPPERCASE`

Standard: Eingeschaltet

Bemerkung: Der Befehl UPPERCASE teilt dem Linker mit, daß alle Symbolnamen vor Ihrer Eintragung in die Symboltabelle in Großbuchstaben umgewandelt werden sollen, was auch der Blinker-Voreinstellung entspricht.

Siehe auch: MIXCASE

VERBOSE

Zweck: Gibt während des Link-Prozesses Status-Informationen aus

Syntax: `VERBOSE [0|1|2]`

Standard: Ausgeschaltet

Beispiel:
```
# erzeugen einer VERBOSE-Ausgabe, welche in die
# Datei TEST.MSG umgeleitet wird
BLINKER @TEST VERBOSE >TEST.MSG
```

Bemerkung: Der Befehl VERBOSE veranlaßt den Linker, Informationen über die augenblicklich vorgenommene Operation auszugeben. Dieser Befehl verlangsamt den Link-Prozeß merklich, gibt dem Entwickler aber die Möglichkeit, die Ursache für einen Linkzeit-Fehler (Fehler beim Linken) zu lokalisieren.

Wurde VERBOSE angegeben, dann zeigt der Blinker jedes bearbeitete Objektmodul sowie die jeweilige Datei an, in der das Modul enthalten ist. Diese Ausgabe kann mit Hilfe der Standardumleitungsbefehle von DOS in eine Datei umgeleitet werden, wenn es erforderlich ist.

Der numerische Parameter des Befehls VERBOSE übergibt die angeforderte Stufe der Informationstiefe:

0 Deaktiviert die VERBOSE-Ausgabe;

1 Gibt während der Bearbeitung die Dateinamen aus;

2 Gibt die Datei- und die Modulnamen aus (Standard).

Ein Beispiel für die über den Befehl VERBOSE generierte Ausgabe finden Sie im Anhang A.

Siehe auch: BLINKER MESSAGE DUPLICATES
MAP

WORKFILE

Zweck: Gibt das Laufwerk, den Pfad und den Dateinamen einer temporären Arbeitsdatei an

Syntax: `WORKFILE [L:][Pfad]<Dateiname>`

Beispiel: `WORKFILE C:\TEMP\Temp`

Bemerkung: Der Befehl WORKFILE gibt dem Linker Laufwerk, Pfad und Namen der zur Linkzeit temporär zu erstellenden Arbeitsdatei an, falls eine solche angelegt werden muß. Standardmäßig zieht Blinker die Verwendung von entweder XMS-(2.0 oder höher) oder EMS-(3.2 oder höher)-Speicher, falls vorhanden, einer Arbeitsdatei vor.

Falls der Befehl WORKFILE nicht angegeben ist und Blinker während des Linklaufs auf die Platte swappen muß, erzeugt er eine Datei mit eindeutigem Namen in dem durch die Umgebungsvariable TEMP oder TMP angegebenen Verzeichnis, falls sie angegeben wurde, oder im aktuellen Verzeichnis, falls sie nicht vorhanden ist.

Blinker-Funktionen zur Programmierung

Funktionsübersicht

Funktions-Namenskonventionen

Funktions-Notationskonventionen

Funktions-Parametertyp-Konventionen

Funktions-Aufruf-Konventionen

Funktions-Kurzübersicht

Allgemeine Funktionsreferenz

DOS-Extender-Funktionsreferenz

SWAP- / RUN-Funktionsreferenz

Funktionsübersicht

Blinker stellt dem Entwickler eine Reihe von Funktionen zur Verfügung, mit denen die Overlay-Umgebung zur Laufzeit kontrolliert und Informationen zum Debugging gewonnen werden können. Alle BLINKER-Funktionen ihres Programmes werdn automatisch gelinkt, ohne daß Sie eine zusätzliche Bibliothek angeben müssen. Blinker-Funktionen können nur in Programmen verwendet werden, die mit Blinker gelinkt sind.

Funktions-Namenskonventionen

Bemerkung für Assembler-Programmierer:

Falls Sie Funktionen von Assembler aus aufrufen, fügen Sie bitte einen Unterstrich '_' vor jeden Funktionsnamen, um mit den C-Namenskonventionen übereinzustimmen.

Bemerkung für BASIC-Programmierer:

Blinker-Funktionen, die von BASIC aufgerufen werden und die einen String als Parameter erhalten oder einen String zurückgeben, erhalten einen neuen Namen: die Buchstaben "BAS" werden an den normalen neunstelligen Funktionsnamen angehängt. Z.B. wird die BASIC-Version von BLIDEMDTE() aufgerufen als BLIDEMDTEBAS().

Funktions-Notationskonventionen

Viele der durch Blinker zur Verfügung gestellten Funktionen können von mehreren Programmiersprachen aufgerufen werden, solange diese die Standard-C-Aufruf-Konventionen erfüllen.

Blinker stellt Schnittstellen für C/C++, Microsoft Basic PDS und MS QuickBasic und CA-Clipper zur Verfügung. Sollten Sie eine hier nicht aufgeführte Programmiersprache verwenden, können Sie wahrscheinlich Ihren Compiler dazu veranlassen, die C-Aufruf-Konventionen zu verwenden, damit Sie auf die Blinker-Funktionen zurückgreifen können. Sehen Sie bitte in Ihr Compiler-Handbuch, um weitere Informationen diesbezüglich zu erhalten.

Die Tabellen auf den folgenden Seiten erläutern die Notations-Konventionen, um die Parametertypen und Rückgabewerte zu beschreiben, sowie die äquivalenten Datentypen für jede Aufruf-Konvention.

Jedem Parameter (oder Rückgabewert) werden ein oder zwei Buchstaben vorangestellt, welche den Typ des Wertes und die Art der Übergabe des Parameters an die Funktion beschreiben.

Hinweis für C- und Assembler-Programmierer: C- und Assembler-Programme, die Blinker-Funktionen aufrufen, sollten für alle nichtverwendeten Parameter den Wert -1 übergeben.

Funktionsparameter-Typkonventionen

Die folgende Tabelle zeigt die Datentypen, die als Parameter und Rückgabewerte von allen Blinkerfunktionen verwendet werden. Die Prefixe und ihre Deklarationen für BASIC, C/C++ und Clipper sind wie folgt:

ni	signed integer	signed word	numeric
nu	unsigned integer	unsigned word	numeric
nl	long integer	signed dword	numeric
c	string	byte pointer	character
l	boolean	word	logical

(in CA-Clipper braucht kein Typ vereinbart werden, jedoch muß der übergebene Parameter vom korrekten Typ sein)

Logische (oder boolsche) Werte werden nur für CA-Clipper direkt unterstützt. Alle anderen Sprachen sollten einen numerischen Integer verwenden, wobei eine Übergabe des Wertes 0 FALSE und eines Wertes ungleich 0 TRUE anzeigt. Bei den Rückgabewerten der Funktonen gilt gleiches.

Nicht aufgeführte Programmiersprachen sollten Datentypen benutzen, die den C-Typen entsprechen.

Funktions-Aufruf-Konventionen

Wegen der Kompatibilität zu anderen Umgebungen beginnen die Blinker DOS Extender Funktionen alle mit dem Wort ‘Dos’ und verwenden die PASCAL Aufrufkonventionen. Die Prototypen für diese Funktionen sind in der Header-Datei BLX286.H im C Unter-verzeichnis des Blinker Installationsverzeichnisses definiert. Diese Funktionen sollten nur von Lowlevel Sprachen wie C oder Assembler aus aufgerufen werden.

Alle übrigen Blinkerfunktionen verwenden die C Aufrufkonven-tionen und verfügen über Schnittstellen zu vielen Sprachen. Diese Schnittstellen sind in den sprachabhängigen Include-Dateien BLINKER.BI, BLINKER.H, BLINKER.CH, BLINKER.FI, BLINKER.HDR, BLINKER.DEF and BLINKER.INC definiert. Diese Dateien befinden sich in den BAS, C, CLP, FOR, FRC, MOD und PAS Unterverzeichnissen des Installationsverzeichnisses von Blinker.

Die CA-Clipper Schnittstelle verwendet das CA-Clipper Extend-System und benötigt keinerlei weitere Vorkehrungen, von Seiten des Programmierers.

Die BASIC-Schnittstelle ist identisch zur C-Schnittstelle für einfache Datentypen, jedoch benötigen Strings eine besondere Behandlung. Aus diesem Grund sind alle Blinker-Funktionen, die unter BASIC eingesetzt werden sollen und Strings bearbeiten, mit einer zusätzlichen Endung ‘BAS’ zu versehen.

Die C-Schnittstelle verwendet die folgenden Aufruf-Konventionen:

- einfache Datentypen werden auf dem Stack abgelegt
- Strings werden als ‘far’-Referenz übergeben
- Parameter werden in umgekehrter Reihenfolge übergeben, d.h.,der letzte Parameter wird als erster auf den Stack geschoben.

Diese Konventionen entsprechen den Aufrufkonventionen von C.

Sprachen, für die keine Schnittstelle vorhanden ist, sollten die C-Schnittstelle verwenden.

Funktions-Kurzübersicht

Die Blinker-Funktionen lassen sich in drei Kategorien einteilen : die allgemeinen Funktionen, die DOS Extender Funktionen und die Swap / Run Funktionen.

Allgemeine Funktionsreferenz

DOS-Extender-Funktionsreferenz

SWAP / RUN-Funktionsreferenz

CA-Clipper spezifische Funktionen

Allen CA-Clipper Funktionen, die Ausgaben erzeugen, kann ein File Handle als Parameter übergeben werden, sodaß die entsprechenden Ausgaben an dieses File Handle geschickt werden können. Die Nummer des File Handles wird nicht überprüft und kann entweder auf 1 zur Ausgabe auf dem Bildschirm (Voreinstellung), 2 zur Ausgabe auf der Standardausgabe, 4 zur Ausgabe auf dem Drucker oder die Handle-Nummer einer mit fopen() geöffneten Datei eingestellt werden.

Das vorgegebene Handle kann zur Linkzeit mit dem Befehl BLINKER DEBUG HANDLE oder in der Applikation mit der Debug-Handle-Funktion BLIDBGHAN() eingestellt werden. Sollte kein Handle angegeben sein, wird die Ausgabe zum DOS-File Handle 1 gesendet (in der Regel der Bildschirm).

Allgemeine Funktionsreferenz

Der Buchstabe C in der ersten Spalte zeigt an, ob eine Funktion nur in einem CA-Clipper Programm zur Verfügung steht, und die Buchstaben D und W in der zweiten und dritten Spalte zeigen an, ob die Funktion in einem DOS oder einem Windows Programm verfügbar ist.

C	D	Blinker Debughandle ermitteln BLIDBGHAN()
	D	Ablaufdatum der Demonstration ermitteln BLIDEMDTE() / BLIDEMDTEBAS()
	D	Ablaufdauer der Demonstration in Minuten ermitteln BLIDEMMIN()
C	D	Speicherfragmentierung anzeigen BLIDISFRG()
	D	BLINKER Fehlernummer ermitteln BLIERRNUM()
	D	BLINKER Fehlerparameter ermitteln BLIERRPRM() / BLIERRPRMBAS()
C	D	Funktionshandle einer DLL Bibliothek aufrufen BLIFUNCAL()
C	D	Funktionshandle einer DLL Bibliothek ermitteln BLIFUNHAN()
C	D	eine DLL Bibliothek freigeben BLILIBFRE()
C	D	eine DLL Bibliothek laden

BLILIBLOD()

C D DLL Bibliotheksfunktionen erlauben, statische Funktionen zu überschreiben
BLILIBOVR()

C D Speicherfragmentierung auflisten
BLILSTFRG()

C D gesamter verfügbarer Speicher ermitteln
BLIMEMAVL()

C D verfügbarer Speicher in Blöcke ermitteln
BLIMEMBLK()

C D maximale Speichergröße eines Blocks ermitteln
BLIMEMMAX()

C D Speicherpackfrequenz ermitteln und setzen
BLIMEMPAK()

C D gesamte Speichergröße ermitteln
BLIMEMSIZ()

C D benutzte Speichergröße ermitteln
BLIMEMUSE()

D Programmstatusinformationen ermitteln
BLIMGRSTS()

C D nicht aktive Overlays löschen
BLIOVLCLR()

D Einstellung für Overlaypool Opsize ermitteln
BLIOVLOPS()

D mit Overlaymanager fortfahren
BLIOVLRES()

D aktuelle Größe des Overlaypools ermitteln
BLIOVLSIZ()

D Overlaymanager anhalten
BLIOVLSUS()

D einen Protected Mode Zeiger dekrementieren
BLIPTRDEC()

D einen Protected Mode Zeiger inkrementieren
BLIPTRINC()

D Seriennummer ermitteln
BLISERNUM() / BLISERNUMBAS()

C D Abbildung der Speicherfragmentierung als String
BLISTRFRG()

D Blinker Versionsnummer ermitteln

DOS-Extender-Funktionsreferenz

Die folgenden Funktionen können für den Blinker DOS Extender benutzt werden. Sie haben selten eine sinnvolle Wirkung und geben auch keine sinnvolle Information zurück, wenn sie nicht mit dem DOS Extender verwendet werden.

In den nachfolgenden Erläuterungen und Beispielen werden folgende Bergiffe gebraucht:

Deskriptor:	Eine Struktur, die die Größe, Basisadresse und Attribute eiens Protected Mode Segments enthält.
Paragraph:	Der Segment-Teil einer Real Mode Adresse, z.B. ist der Paragraph des Farbdisplay Adapters B800h.
Segment:	Die eigentliche Code- oder Dateneinheit, wie sie im Speicher reserviert und belegt ist.
Selector:	Der Wert, mit dem man ein Segment im Protected Mode identifizieren kann, d.h., der an Protected Mode Speicherallokierungsfunktionen übergeben oder von ihnen zurückgegeben wird.

Alle Funktionen erwarten die folgenden Definitionen aus der Include-Datei BLX286.H, die im C Unterverzeichnis von Blinker zu finden ist.:

```
typedef unsigned char BYTE;
typedef unsigned short SEL;
typedef unsigned short USHORT;
typedef unsigned long ULONG;
typedef unsigned short _far *PSEL;

typedef struct
{
   ULONG segbase;       /* Segment lineare Basisadresse */
   ULONG seglen;    /* Segmentlänge in Bytes */
   USHORT segattrib;   /* Segment Attribut-Bytes */
}DESC;              /* Das ist kein Real Deskriptor */

#define BLXAPI pascal _far
```

Es gibt auch einige C Beispielprogramme, die die Anwendung der DOS Extenderfunktionen zeigen. Sie befinden sich ebenfalls im C Unterverzeichnis von Blinker.

Belegt einen Block des Erweite rungsspeichers
DosAllocHuge()

Belegt einen Block des konventionellen Speichers
DosAllocRealSeg()

Belegt einen Block des Erweiterungsspeichers

__DosAllocSeg()

Erzeugt einen ausführbaren Selektor der einem Datenselektor entspricht
__DosCreateCSAlias()

Erzeugt einen Datenselektor, der einem ausführbaren Selektor entspricht
__DosCreateDSAlias()

Gibt einen Selektor für ein Real Mode Segment frei
__DosFreeRealSeg()

Gibt einen Selektor für ein Protected Mode Segment frei
__DosFreeSeg()

Gibt einen mapped oder aliased Selektor frei
__DosFreeSelector()

Gibt einen Selektor für den BIOS-Datenbereich zurück
__DosGetBIOSSeg()

Ermittelt die Protected Mode-Adresse eines Exception-Handler
__DosGetExceptionHandler()

Ermittelt den aktuellen DOS-Extender-Host-Modus
__DosGetHostMode()

Ermittelt den Shift für einen Huge-Inkrementwert
__DosGetHugeShift()

Gibt den aktuellen Maschinenmodus zurück
__DosGetMachineMode()

Gibt einen Protected Mode-Zeiger auf einen Protected Mode-Interrupt-Handler zurück
__DosGetProtVec()

Gibt Real und Protected Mode-Zeiger auf Real und Protected Mode-Interrupt-Handler zurück
__DosGetRealProtVec()

Gibt einen Real Mode-Zeiger auf einen Real Mode-Interrupt-Handler zurück
__DosGetRealVec()

Gibt die Basisadresse eines Selectors zurück
__DosGetSegBase()

Gibt die Deskriptorstruktur eiens Selectors zurück
__DosGetSegDesc()

Gibt die Größenbegrenzung eines Selektors zurück
__DosGetSegLimit()

Ermittelt, ob der Blinker DOS-Extender verwendet wird
__DosIsBlinkX()

Sperrt ein Segment gegen Auslagerung aus Speicher
__DosLockSeg()

Kovertiert eine Real Mode Paragraphenadresse in einen Protected Mode Selector
__DosMapRealSeg()

Ermittelt den gesamten verfügbaren Extended Speicher
__DosMemAvail()

Gibt einen Real Mode Zeiger zurück, der einem Protected Mode Zeiger entspricht
__DosProtToReal()

Ermittelt den gesamten verfügbaren konventionellen Speicher
__DosRealAvail()

Erzeugt einen Far-Aufruf auf eine Real Mode-Adresse
__DosRealFarCall()

Erzeugt einen Real Mode Interrupt
__DosRealIntr()

Resize a huge extended memory block
__DosReallocHuge()

Verändert die Größe eines erweiterten Speicherblocks
__DosReallocSeg()

Gibt einen Protected Mode Zeiger zurück, der einem Real Mode Zeiger entspricht
__DosRealToProt()

Setzt die Protected Mode-Adresse eines Exception-Handler
__DosSetExceptionHandler()

Installiert einen Interrupt-Handler für den Protected Mode
__DosSetProtVec()

Installiert einen Interrupt-Handler für den Real Mode
__DosSetRealVec()

Setzt die Segmentattribute für einen Selektor
__DosSetSegAttrib()

Entsperrt ein Segment, das zuvor über DosLockSeg() gesperrt worden ist
__DosUnlockSeg()

Ermittelt den verfügbaren virtuellen Speicher
__DosVMAvail()

Bestimmt die Flags für einen Selector
__DosVerifyAccess()

SWAP / RUN-Funktionsreferenz

Die SWAP / RUN - Funktionen verwenden die C-Aufruf-konventionen, wie sie im Abschnitt ‘Funktions-Aufruf-Konventionen’ beschrieben sind.

Die Schnittstellen der SWAP / RUN – Funktionen zu Clipper, Basic, C etc. werden in den sprachspezifischen Include-Dateien definiert. Diese Dateien, BLINKER.BI, BLINKER.H, BLINKER.CH, BLINKER.FI, BLINKER.HDR, BLINKER.DEF und BLINKER.INC, befinden sich im entsprechenden BAS, C, CLP, FOR, FRC, MOD und PAS Unterverzeichnis von Blinker.

Erhöht die Größe der Umgebung für den untergeordneten Prozeß
SWPADDENV()

Gibt einen String an die übergeordnete Anwendung zurück
SWPADDSTR() / SWPADDSTRBAS()

Speichern/Laden des aktuellen Verzeichnisses nach einer Auslagerung
SWPCURDIR()

Eine Nachricht nach einer Auslagerung anzeigen
SWPDISMSG()

Ermittelt den Rückgabewert des Kind-Prozesses
SWPERRLEV()

Haupt-Fehlercode der Auslagerung ermitteln
SWPERRMAJ()

Unter-Fehlercode der Auslagerung ermitteln
SWPERRMIN()

Gibt an, wieviel EMS-Speicher freibleiben soll
SWPFREEMS()

Gibt an, wieviel XMS-Speicher freibleiben soll
SWPFREXMS()

Auf einen Tastendruck nach einer Auslagerung warten
SWPGETKEY()

die aktuelle Programm-ID ermitteln
SWPGETPID() / SWPGETPIDBAS()

Gibt den Rückgabestring des unter-geordneten Prozesses zurück
SWPGETSTR() / SWPGETSTRBAS()

Übergibt Tastendrücke an ein untergeordnetes Programm
SWPKEYBRD() / SWPKEYBRDBAS()

Unterbindet die Löschung des Tastaturpuffers
SWPKEYCLR()

Reboot während der Auslagerung ausschalten
__SWPNOBOOT()

Einen Kind-Prozess ausführen
__SWPRUNCMD() / SWPRUNCMDBAS()

Umgebungsvariablen für den untergeordneten Prozeß setzen
__SWPSETENV() / SWPSETENVBAS()

Aktuelle Programm-ID setzen
__SWPSETPID() / SWPSETPIDBAS()

Einen String an das übergeordnete Programm übergeben
__SWPSETSTR() / SWPSETSTRBAS()

EMS für die Auslagerung verwenden
__SWPUSEEMS()

UMBs für die Auslagerung verwenden
__SWPUSEUMB()

XMS für die Auslagerung verwenden
__SWPUSEXMS()

Sichern/Wiederherstellen des Video-Modus
__SWPVIDMDE()

BLIDBGHAN()

Funktion: Setzen und Lesen des File-Handles für die Debug-Ausgabe

Syntax: `nuHandleAlt = BLIDBGHAN([nuHandleNeu])`

Parameter: <nuHandleNeu>= File-Handle für Debug-Ausgabe

Rückgabe: Vorheriger Debug-Handle

Bemerkung: Diese Funktion setzt den Debug Handle des BLINKER-Overlay-Managers. Sie gibt auch den bisherigen Debug Handle zurück.

Der Handle kann zur Laufzeit mit der Einstellung /DHn in der BLINKER-Umgebungsvariablen überschrieben werden.

Die Nummer des Handle wird nicht überprüft und kann entweder auf 1 zur Ausgabe auf den Bildschirm (Voreinstellung), 2 zur Ausgabe auf die Standard-Fehlereinheit (standard error device) oder 4 zur Ausgabe auf den Drucker oder auf die Handle-Nummer einer mit FOPEN() geöffneten Datei eingestellt werden.

Dieser Handle wird standardmäßig als Ausgabe-Handle für andere Blinker Debugging-Funktionen verwendet. Daher sollte der Handle schon am Anfang des Programms gesetzt werden, anstatt ihn als Parameter an jede einzelne Funktion zu übergeben.

Beispiel:

```
#include <stdio.h>
#include <blinker.h>

void main(void)
{
printf ("Der Debug Handle war %i und ist nun der Drucker\n",
  BLIDBGHAN(4));
}
```

Siehe auch: BLINKER DEBUG HANDLE

BLIDEMDTE / BLIDEMDTEBAS()

Funktion: Demo-Ablaufdatum ermitteln, falls gesetzt.

Syntax: `cWert = BLIDEMDTE()`

Parameter: keine

Rückgabe: Ablaufdatum als String

Bemerkung: Diese Funktion gibt das Ablaufdatumdatum für die Demo-Version einer Applikation, falls während des Linkens gesetzt, im Fomat JJJJMMTT zurück.

Falls ein Demo-Ablaufdatum nicht festgesetzt worden ist, wird ein Nullstring zurückgegeben.

Diese Funktion wird zur Zeit nur von DOS und DOS Extended Programmen unterstützt, jedoch nicht von Windowsprogrammen.

Beispiel:

```
#include <stdio.h>
 #include <blinker.h>

 void main(void)
 {
 char demodate [9];
 strcpy (demodate,BLIDEMDTE());
 if (strlen(demodate) > 0)
   printf ("Dieses Demoprogramm wird ablaufen am %s\n",demodate);
 }
```

Siehe auch: BLINKER DEMONSTRATION DATE

BLIDEMMIN()

Funktion: Demo-Zeitlimit in Minuten setzen und auslesen

Syntax: `nuDemoMinuten = BLIDEMMIN([nuDemoMinuten])`

Parameter: <nuDemoMinuten> : Demo-Zeitlimit in Minuten

Rückgabe: Demo-Zeitlimit in Minuten

Bemerkung: Diese Funktion gibt die maximale Laufzeit für die Demo-Version einer Applikation in Minuten zurück, soweit eine solche festgesetzt worden ist. Der Rückgabewert ist eine Integerzahl im Bereich von 0 (= keine Zeitbeschränkung gesetzt) bis 65535.

Diese Funktion kann außerdem zur Erweiterung des bisherigen Zeitlimits eingesetzt werden, indem eine neue Grenze als Parameter übergeben wird. Eine solche Erweiterung kann z.B. dann sinnvoll sein, wenn man den augenblicklichen Programmteil nicht abbrechen, sondern zu einem normalen Ende führen möchte. Die neue Zeitbegrenzung sollte naturgemäß größer als der von BLIDEMMIN() zurückgegebene Wert sein.

Beispiel:

```
#include <stdio.h>
```

```
#include <blinker.h>

void main(void)
{
int demominutes;
demominutes = BLIDEMMIN(0);
if (demominutes > 0)
  {
  printf ("Dieses Programm wird in %d Minuten enden\n",
     demominutes);
  printf ("Die Laufzeit wird um 2 Minuten verlängert\n");
  BLIDEMMIN(demominutes+2);
  }
}
```

Siehe auch: BLINKER DEMONSTRATION MINUTES

BLIDISFRG()

Funktion: Anzeige der Fragmentierung des freien Speicherpools für CA-Clipper Sommer '87

Syntax: `BLIDISFRG([nuHandle])`

Parameter: <nuHandle> : Handle für Ausgabe

Rückgabe: Keine

Bemerkung: Diese Funktion gibt eine zusammengefaßte Darstellung des dynamischen Speichers von CA Clipper Sommer '87 aus, um die Fragmentierung aufzuzeigen. Als optionaler Parameter kann ein Handle für die Ausgabe angegeben werden. Jede Zeichenposition repräsentiert dabei bis zu 512 Bytes und aufeinanderfolgende gleiche Zeichen stellen Blöcke von mehr als 512 Bytes dar.

Es werden zwei Speicherbereiche mit dieser Funktion angezeigt; der größte Bereich ist der 'far heap', auch bezeichnet als FREE POOL (freier Bereich), indem die meisten Speicherbelegungen durch CA-Clipper ausgeführt werden, und der 'near heap', ein wesentlich kleinerer Block zusammenhängenden Speichers (maximal 64KByte), welcher durch CA-Clipper immer dann verwendet wird, wenn die Belegung auf dem FREE POOL nicht erfolgreich ausgeführt werden konnte.

Das Zeichen * und * stellen freie Speicherblöcke dar, die nicht von CA-Clipper benutzt werden. Die Grafikzeichen * und * stellen Speicherblöcke dar, die von CA-Clipper belegt sind. Das Zeichen * zeigt die Stelle an, wo die nächste Speierreservierung erfolgen wird. Die Bedeutung der Zeichen wird unterhalb der Speicher-Übersicht angezeigt.

Der Grund für die Verwendung von je zwei unterschiedlichen Zeichen für die freien und die belegten Blöcke liegt in der übersichtlicheren Anzeige, wenn ein freier oder belegter Bereich aus mehreren Blöcken besteht. Bei Verwendung von nur einem Zeichen könnte nicht das volle Ausmaß der Speicher-Fragmentierung in sehr kleine Blöcke dargestellt werden.

Die relativen Größen der Blöcke werden durch identische, aufeinanderfolgende Zeichen

angezeigt, da ein einzelnes Zeichen einen Speicherblock kleiner als 512 Bytes und eine Reihe von 4 Zeichen ungefähr einen 2 KByte Block darstellt. Die relative Fragmentierung wird durch das Verhältnis von freien zu belegten Blöcken und den Wechsel zwischen freien und belegten Blöcken dargestellt.

Wenn freie Bereiche einmal durch belegte Bereiche unterbrochen sind, werden sie fast unbrauchbar, außer für die Speicherung von kleinen temporären Werten. Daher nimmt der freie Speicher (MEMORY(0)) immer mehr ab und Operationen wie MEMOEDIT() oder eine INDIZIERUNG, die größere Speicherblöcke benötigen, schlagen fehl.

Diese Funktion kann während der Ausführung einer Applikation zur Beobachtung der Fragmentierung von Speicher benutzt werden. Außerdem können die Verbesserungen, die durch die Funktion BLIMEMPAK() und den Befehl BLINKER MEMORY PACK erreicht werden, direkt geprüft werden.

Beachten Sie, daß CA-Clipper Sommer '87 den FREE POOL 'bei Bedarf' vom verfügbaren DOS-Speicher belegt. Solange nicht MEMORY(0), BLIMEMPAK(-1) oder BLIMEMBLK() ausgeführt werden (was CA-Clipper dazu veranlaßt, allen freien DOS-Speicher dem FREE POOL hinzuzufügen), zeigt diese Funktion nur den aktuell belegten FREE POOL an.

Beispiel:
```
//  Anzeige der Speicher-Fragmentierung:
//
BLIDISFRG(4)// die Ausgabe wird zum Drucker gesendet
```

Siehe auch: BLILSTFRG()
BLISTRFRG()
BLINKER MEMORY PACK

BLIERRNUM()

Funktion: BLINKER-Fehlernummer ermitteln

Syntax:
```
nuWert = BLIERRNUM()
```

Parameter: keine

Rückgabe: Blinker-Fehlernummer

Bemerkung: Diese Funktion gibt die Nummer des Blinker-Laufzeitfehlers (runtime-error) zurück und dient zum Einsatz in der BLINKER-Fehlerbehandlungs-Prozedur (Error Handler) BLERRPRG. Bei normaler Ausführung wird der Wert Null zurückgegeben, d.h. es liegt kein Fehler vor.

Eine ausführliche Erläuterung des Error-Handlers von BLINKER finden Sie in Kapitel 7 "Spezielle Features"

Eine ausführliche Erläuterung der Laufzeitfehler von BLINKER finden Sie in Kapitel 11 "Fehler und Fehlermeldungen".

Beispiel:
```
//  Ermittle die Blinker-Fehlernummer:
//
? "Die BLINKER-Fehlernummer ist ",BLIERRNUM()
```

Siehe auch: BLIERRPRM() / BLIERRPRMBAS()

BLIERRPRM / BLIERRPRMBAS()

Funktion: BLINKER-Fehlerparameter ermitteln

Syntax:
```
cWert = BLIERRPRM()
```

Parameter: kein

Rückgabe: String mit dem Dateinamen, der zu dem Blinker Fehler gehört.

Bemerkung: Einige der Laufzeitfehler von BLINKER beziehen sich auf eine bestimmte, den Fehler verursachende Datei. Diese Funktion liefert den Namen der entsprechenden Datei, z.B. zur Anzeige in einer Fehlermeldung.

Eine ausführliche Erläuterung des Error-Handlers von BLINKER finden Sie in Kapitel 7 "Spezielle Features".

Eine ausführliche Erläuterung der Laufzeitfehler von BLINKER finden Sie in Kapitel 11 "Fehler und Fehlermeldungen".

Beispiel:
```
//  Ermittle den BLINKER-Fehlerparameter:
//
? "Die fehlerverursachende Datei heißt : ", BLIERRPRM()
```

Siehe auch: BLIERRNUM()

BLIFUNCAL()

Funktion: Aufruf einer CA-Clipper DLL-Bibliotheksfunktion

Syntax:
```
Ergebnis = BLIFUNCAL(param1,. . .,nuHandle)
```

Parameter: param1. . . Funktions-Parameter.

NuHandle Funktions-Handle aus BLIFUNHAN().

Rückgabe: Ergebnis der DLL-Bibliotheksfunktion.

Bemerkung: Diese Funktion wird zum Aufruf einer CA-Clipper-Funktion in einer DLL verwendet, die zur Laufzeit dynamisch geladen wurde.

Nach dem dynamischen Laden einer DLL wird die Funktion BLIFUNHAN() aufgerufen, um den Handle der Funktion über den Funktionsnamen aus der DLL zu erhalten. Der Funktions-Handle bleibt bis zur Freigabe der DLL über BLILIBFRE() gültig.

Um die DLL-Funktion aufzurufen, werden die Parameter zusammen mit dem Funktions-Handle an BLIFUNCAL() übergeben, wobei der Handle als letzter Parameter angegeben wird. BLIFUNCAL() liefert den Rückgabewert der aufgerufenen Funktion zurück.

Beispiel:
```
libhan = BLILIBLOD (dllnme)             // Dynamisches Laden der DLL
if libhan > 32                          // erfolgreich geladen?
  funhan = BLIFUNHAN (libhan,funnme)    // Nr. des Handles
  if funhan <> 0                        // Funktion gefunden?
```

```
                    // Funktionsaufruf mit (mehreren) Parametern
      funres = BLIFUNCAL (funpa1,funpa2,funhan)
                    // Hinweis: Handle ist letzter Parameter
  endif
  BLILIBFRE(libhan)                  // Erledigt, also freigeben
endif
```

Hinweis: Siehe auch Kapitel 5, 'DOS-Extender-/Dual Mode-Programme' für mehr Einzelheiten zur Verwendung von DLLs.

Siehe auch: BLIFUNHAN()
BLILIBFRE()
BLILIBLOD()
BLILIBOVR()

BLIFUNHAN()

Funktion: Den Handle einer CA-Clipper DLL-Bibliotheksfunktion ermitteln.

Syntax: `NuFunHandle = BLIFUNHAN(nuLibHandle,cFunName)`

Parameter: nuLibHandle Handle der DLL-Bibliothek

cFunName Name der gewünschten Funktion

Rückgabe: Die Nummer des Handle für die Funktion oder Null, falls die Funktion in der DLL nicht existiert.

Bemerkung: Diese Funktion wird eingesetzt, um den Handle für eine CA-Clipper-DLL-Funktion aus der DLL-Bibliothek und dem Funktionsnamen zu erhalten. Dieser Funktions-Handle wird zum Aufruf der Funktion verwendet und bleibt gültig, bis die DLL über BLILIBFRE() freigegeben wird.

Um die DLL-Funktion aufzurufen, werden die Aufrufparameter zuzüglich der Handle-Nummer als letztem Parameter an die Funktion BLIFUNCAL() übergeben.

Beispiel:

```
libhan = BLILIBLOD (dllnme)             // DLL dynamisch laden
if libhan > 32                          // erfolgreich geladen?
  funhan = BLIFUNHAN (libhan,funnme)    // Nr. des Handles
  if funhan <> 0                        // Funktion gefunden?
                    // Funktionsaufruf mit (mehreren) Parametern
      funres = BLIFUNCAL (funpa1,funpa2,funhan)
                    // Hinweis: Funktions-Handle letzter Parameter
  endif
  BLILIBFRE(libhan)                     // Erledigt, also freigeben
endif
```

Hinweis: Siehe auch Kapitel 5, 'DOS-Extender-/Dual Mode-Programme' für mehr Einzelheiten zur Verwendung von DLLs.

Siehe auch: BLIFUNCAL()
BLILIBFRE()
BLILIBLOD()
BLILIBOVR()

BLILIBFRE()

Funktion: CA-Clipper DLL-Bibliothek freigeben.

Syntax: `BLILIBFRE(nuHandle)`

Parameter: nuHandle DLL-Bibliotheks-Handle.

Rückgabe: Keine.

Bemerkung: Diese Funktion gibt eine dynamisch geladene CA-Clipper-DLL frei, wenn sie nicht mehr benötigt wird.

Alle Funktions-Handles, die einer Bibliothek über BLIFUNHAN() zugewiesen wurden, werden bei Freigabe der Bibliothek ungültig.

Beispiel:
```
libhan = BLILIBLOD (dllnme)             // DLL dynamisch laden
 if libhan > 32                         // Laden erfolgreich?
   funhan = BLIFUNHAN (libhan,funnme)   // Nr. des Handles
   if funhan <> 0                       // Funktion gefunden?
                    // Funktionsaufruf mit (mehreren) Parametern
      funres = BLIFUNCAL (funpa1,funpa2,funhan)
                    // Hinweis: Funktions-Handle letzter Parameter
   endif
   BLILIBFRE(libhan)                    // Erledigt, also freigeben
 endif
```

Hinweis: Diese Funktion wird hauptsächlich wegen der Konsistenz angeboten. Denn der Speicher und andere Resourcen, die von einem CA-Clipper .DLL benutzt werden, können nicht wirklich freigegeben werden, bevor die Anwendung endet. Dies liegt an der Art und Weise, wie die Laufzeitsymboltabelle von Clipper verwaltet wird.

Sehen Sie auch Kapitel 5, ‘DOS-Extender-/Dual Mode-Programme’ für mehr Einzelheiten zur Verwendung von DLLs.

Siehe auch: BLIFUNCAL()
BLIFUNHAN()
BLILIBLOD()
BLILIBOVR()

BLILIBLOD()

Funktion: Dynamisches Laden einer CA-Clipper-DLL.

Syntax: `NuHandle = BLILIBLOD(cLibName)`

Parameter: cLibName Name der DLL-Bibliothek

Rückgabe: Der Handle für die DLL oder eine Fehlernummer kleiner 32, falls das Laden nicht erfolgreich war.

Bemerkung: Über diese Funktion wird eine CA-Clipper-DLL zur Laufzeit dynamisch geladen. Dadurch ist eine Aktualisierung oder Erweiterung von Programmen auch ohne neues Linken der .EXE-Datei möglich.

Der Bibliotheks-Handle wird eingesetzt, um die Funktions-Handles aus der DLL zu erhalten, über die dann die Funktionsaufrufe erfolgen können.

Beispiel:
```
libhan = BLILIBLOD (dllnme)             // DLL dynamisch laden
 if libhan > 32                         // erfolgreich geladen?
   Funhan = BLIFUNHAN (libhan,funnme)   // Nr. des Handles
   if funhan > 0                     // Funktion gefunden?
                    // Funktionsaufruf mit (mehreren) Parametern
      funres = BLIFUNCAL (funpa1,funpa2,funhan)
                    // Hinweis: Funktions-Handle letzter Parameter
   endif
   BLILIBFRE(libhan)                    // Erledigt, also freigeben
 endif
```

Hinweis: Sehen Sie auch Kapitel 5, ‘DOS-Extender-/Dual Mode-Programme’ für mehr Einzelheiten zur Verwendung von DLLs.

Siehe auch: BLIFUNCAL()
BLIFUNHAN()
BLILIBFRE()
BLILIBOVR()

BLILIBOVR()

Funktion: Erzwingt das dynamische Laden von DLL-Bibliotheksfunktionen bei gleichnamigen Funktionen in der .EXE-Datei.

Syntax: `BLILIBOVR(lWert)`

Parameter: lWert Ein-/Ausschalter.

Rückgabe: Keine.

Bemerkung: Über diese Funktion erhalten CA-Clipper-Funktionen aus einer dynamisch geladenen DLL

Vorrang vor gleichnamigen Funktionen in der .EXE-Datei, wodurch existierende Funktionen durch neuere ersetzt werden können.

Standardmäßig verhält sich CA-Clipper so, daß Funktionen in der .EXE-Datei Priorität gegenüber gleichnamigen Funktionen in DLLs besitzen.

Diese Funktion wird typischerweise einmal beim Programmstart aufgerufen, kann aber auch für die Änderung individueller DLLs nach Bedarf eingesetzt werden.

Beispiel:

```
BLILIBOVR(.T.)                              // Erzwingt Vorrang der
                                              // DLL vor EXE
libhan = BLILIBLOD (dllnme)                 // DLL dynamisch laden
if libhan > 32                              // Laden erfolgreich?
  Funhan = BLIFUNHAN (libhan,funnme)        // Nr. des Handles
  if funhan > 0                        // Funktion gefunden?
                    // Funktionsaufruf mit (mehreren) Parametern
     funres = BLIFUNCAL (funpa1,funpa2,funhan)
                    // Hinweis: Funktions-Handle letzter Parameter
  endif
  BLILIBFRE(libhan)                         // Erledigt, also freigeben
endif
```

Hinweis: Sehen Sie auch Kapitel 5, ‘DOS-Extender-/Dual Mode-Programme’ für mehr Einzelheiten zur Verwendung von DLLs.

Siehe auch: BLIFUNCAL()
BLIFUNHAN()
BLILIBFRE()
BLILIBLOD()

BLILSTFRG()

Funktion: Auflisten der Speicherblöcke von CA-Clipper Sommer ’87

Syntax: `BLILSTFRG([nuHandle])`

Parameter: nuHandle : Handle für Ausgabe

Rückgabe: Keine

Bemerkung: Diese Funktion listet jeden Block des dynamischen Speichers von CA-Clipper auf und zeigt den Grad der Fragmentierung. Dabei wird jeder Block in einer separaten Zeile ausgegeben, welche Informationen enthält zur Adresse im Speicher, seine Größe und ob er sich in Benutzung befindet oder nicht. Die Funktion übernimmt als optionalen Parameter das File Handle der Ausgabe.

Beachten Sie, daß CA-Clipper Sommer ’87 den Speicher ‘bei Bedarf’ vom verfügbaren DOS-Speicher belegt. Solange nicht MEMORY(0), BLIMEMPAK(-1) oder BLIMEMBLK() ausgeführt werden (was CA-Clipper dazu veranlaßt, allen freien DOS-Speicher dem FREE POOL hinzuzufügen), zeigt diese Funktion nur den aktuell belegten FREE POOL an.

Beispiel:
```
*
*  Anzeige der Speicher-Fragmentierung:
*
BLILSTFRG(4) && ie Ausgabe wird zum Drucker gesendet
```

Ein Beispiel für eine Ausgabe wäre:

```
Adress          Size      In Use
8198:0000       00018     Y
8198:0014       00184     N
8198:00EA       00058     Y
8198:0120       60020     N
```

Siehe auch: BLIDISFRG()
BLISTRFRG()

BLIMEMAVL()

Funktion: Gesamter freier dynamischer Speicher von CA-Clipper ermitteln

Syntax: `nlWert = BLIMEMAVL()`

Parameter: keine

Rückgabe: Gesamter freier Speicher in Bytes

Bemerkung: Diese Funktion gibt den gesamten, verfügbaren dynamischen Speicher in Bytes zurück. Als dynamischer Speicher wird der von CA-Clipper benutzte Speicher bezeichnet, in dem alle temporären Werte von Variablen und Puffern während der Laufzeit der Applikation abgelegt werden.

Beachten Sie, daß CA-Clipper Sommer ’87 den Speicher ‘bei Bedarf’ vom verfügbaren

DOS-Speicher belegt. Solange nicht MEMORY(0), BLIMEMPAK(-1) oder BLIMEMBLK() ausgeführt werden (was CA-Clipper dazu veranlaßt allen freien DOS-Speicher dem FREE POOL hinzuzufügen), zeigt diese Funktion nur den aktuell belegten FREE POOL an.

Beispiel:

```
*  Ermittle gesamten verfügbaren Speicher :
*
?  "In CA-Clippers freiem Speicher-Pool sind ", BLIMEMAVL()
?? " Bytes verfügbar."
```

Siehe auch: BLIMEMUSE()
BLIMEMMAX()

BLIMEMBLK()

Funktion: Verfügbarer dynamischer Speicher von CA-Clipper Sommer ’87 mit Blöcken von mindestens nuBlockgröße ermitteln.

Syntax: `nuWert = BLIMEMBLK([nuBlockgröße])`

Parameter: nuBlockgrö ße : Blockgröße in KByte

Rückgabe: Freier Speicher in KByte mit Blöcken, die >= [nuBlockgröße] sind

Bemerkung: Diese Funktion gibt den gesamten, verfügbaren dynamischen Speicher in KBytes zurück, der in Blöcken verfügbar ist, die mindestens so groß sind, wie der Parameter in Kbyte angibt. Als dynamischer Speicher wird der von CA-Clipper benutzte Speicher bezeichnet, in dem alle temporären Werte von Variablen und Puffern während der Laufzeit der Applikation abgelegt werden.

Wird kein Parameter übergeben, so wird eine vorgegebene Blockgröße von 4 KByte benutzt, die den von der CA-Clipper-Funktion MEMORY(0) zurückgegebenen Wert simuliert. BLIMEMBLK() emuliert in allen Fällen die Vorgehensweise von MEMORY(0). Es werden zuerst Blöcke von 63 KByte, dann von 48 KByte, 32 KByte, 16 KByte, 8 KByte, 4 KByte, 2 KByte und schließlich von 1 KBYTE belegt, bis kein Block mehr belegt werden kann. der größer oder gleich dem übergebenen Parameter oder der Voreinstellung ist. Der Rückgabewert ist die Gesamtgröße aller erfolgreich belegten Blöcke.

BLIMEMBLK() ist um einiges schneller als MEMORY(0) und kann daher zur Überwachung des Speichers verwendet werden, wenn MEMORY(0) die Applikation zu stark verlangsamen würde, z.B. beim Indizieren oder Profiling.

Beispiel:

```
* Ermittle den verfügbaren Speicher mit
 * einer Blockgröße von mindestens 8 Kbyte:
 *
 ? " Der verfügbare Speicher mit mind. 8 KByte Blockgröße"
 ? " beträgt ",BLIMEMBLK(8), " KByte. "
```

BLIMEMMAX()

Funktion: Größe des größten verfügbaren CA-Clipper Speicherblocks

Syntax: `nlWert = BLIMEMMAX()`

Parameter: keine

Rückgabe: Größe des größten freien Speicherblocks in Bytes.

Bemerkung: Diese Funktion gibt die Größe des größten verfügbaren Blockes an dynamischen Speicher in Bytes zurück. Als dynamischer Speicher wird der von CA-Clipper benutzte Speicher bezeichnet, in dem alle temporären Werte von Variablen und Puffern während der Laufzeit einer Applikation abgelegt werden.

Dies kann dazu benutzt werden, um zum Beispiel vor einem Aufruf der Funktion MEMOEDIT() zu prüfen, ob ein genügend großer Block freien Speichers zur Verfügung

steht.

Beispiel:

```
* Ermittle die Größe des größten verfügbaren
 * Blocks
 *
 ? "Der größte verfügbare Block ist ",BLIMEMMAX()," Bytes groß. "
```

BLIMEMPAK()

Funktion: CA-Clipper Sommer '87 Speicherpool packen

Syntax: `nuWert = BLIMEMPAK([nuFrequenz | -1])`

Parameter: nuFrequenz : Häufigkeit der Speicherkomprimierung

Rückgabe: Häufigkeit der Speicherkomprimierung

Bemerkung: Diese Blinker-Funktion ermittelt die Häufigkeit der Prozeduraufrufe, mit welchen der Overlay-Manager den freien dynamischen Speicher von CA-Clipper komprimiert. Weiterhin kann ein optionaler Parameter übergeben werden, der die neue Komprimierungsfrequenz einstellt, wobei auch hier die bisherige Komprimierungshäufigkeit als Integerwert zurückgegeben wird.

Der Vorgabewert durch Blinker ist 0, das heißt, daß keine Komprimierung erfolgt und die Applikation in der maximalen Geschwindikeit abläuft. In diesem Fall erfolgt keine Komprimierung, um die Speicherfragmentierung zu reduzieren. Sie können BLINKER mitteilen, daß der dynamische Speicher automatisch nach <nuFrequenz> Prozeduraufrufen komprimiert werden soll. Dies erreichen Sie entweder zur Linkzeit mit dem Befehl BLINKER MEMORY PACK (Siehe Kapitel 9) oder zur Laufzeit der Applikation mit der Funktion BLIMEMPAK().

Der Effekt dieser Funktion hängt vom übergebenen Parameter ab. Es wird aber immer die aktuelle Komprimierungsfrequenz zurückgegeben, d.h., daß der BLINKER-Overlay-Manager jedesmal nach <nuFrequenz> Aufrufen von Overlay-Prozeduren den dynamischen Speicher komprimiert.

Falls kein Parameter übergeben wird, erfolgt durch die Funktion keine Aktion.

Wurde der Wert -1 als Parameter übergeben, dann führt der Overlay-Manager sofort eine Speicherkomprimierung durch und läßt die Komprimierungshäufigkeit unverändert. Das ist insbesondere nach einem Aufruf der Funktion BLIOVLCLR() sinnvoll, um vor DBEDIT, MEMOEDIT, Reindexierung und anderen speicherintensiven Routinen inaktive Overlays zu entfernen und dadurch Speicherreserven freizulegen.

Ein Parameter 0 schaltet die Speicherkomprimierung völlig aus. Alle anderen Werte setzen die Komprimierungshäufigkeit auf den übergebenen Wert.

Falls das automatische Komprimieren abgeschaltet ist, durch Unterlassung von BLINKER MEMORY PACK zur Linkzeit, dann kann ein Aufruf von BLIMEMPAK(-1) beim Start der Applikation eine deutliche Verbesserung der Speicherfragmentierung mit sich bringen. Der Grund dafür ist, daß schon zum Zeitpunkt der ersten CA-Clipper-Anweisung eine ganze Menge an Ladevorgängen und Speicherbelegungen bzw. Freigaben vonstatten gegangen sind, so daß zu Beginn der CA-Clipper-Applikation der Speicher schon telweise

fragmentiert sein kann.

Beispiel:
```
* Ermittle die Speicher-Packhäufigkeit von BLINKER:
*
? "Derzeit wird nach jeweils ",BLIMEMPACK(),"Aufrufen gepackt"
*
* Einstellen der Packrate auf jeden zehnten
* Prozedur-Aufruf
*
BLIMEMPAK(10)
```

Siehe auch: BLINKER MEMORY PACK
BLIOVLCLR()

BLIMEMSIZ()

Funktion: Gesamtgröße des dynamischen Speichers von CA-Clipper Sommer '87 ermitteln.

Syntax: `nlWert = BLIMEMSIZ()`

Parameter: Kein

Rückgabe: Gesamtgröße des FREE POOL in Bytes

Bemerkung: Diese Funktion gibt den gesamten dynamischen Speicher in Bytes zurück. Als dynamischer Speicher wird der von CA-Clipper benutzte Speicher bezeichnet, in dem alle temporären Werte von Variablen und Puffern während der Laufzeit einer Applikation abgelegt werden.

Beachten Sie, daß CA-Clipper Sommer ’87 den Speicher ‘bei Bedarf’ vom verfügbaren DOS-Speicher belegt. Solange nicht MEMORY(0), BLIMEMPAK(-1) oder BLIMEMBLK() ausgeführt werden (was CA-Clipper dazu veranlaßt allen freien DOS-Speicher dem FREE POOL hinzuzufügen), zeigt diese Funktion nur den aktuell belegten FREE POOL an.

Beispiel:
```
*  Ermittle gesamten Speicher:
*
? "CA-Clippers freier Speicher-Pool ist "
?? BLIMEMSIZE()," groß. "
```

Siehe auch: BLIMEMUSE()
BLIMEMAVL()

BLIMEMUSE()

Funktion: Benutzter Speicher im CA-Clipper Sommer '87 Speicherbereich.

Syntax: `nlWert = BLIMEMUSE()`

Rückgabe: Summe der Größe aller Speicherblöcke, die als belegt markiert sind.

Parameter: kein

Bemerkung: Diese Funktion gibt den gesamten, derzeit benutzten dynamischen Speicher in Bytes zurück. Als dynamischer Speicher wird der von CA-Clipper benutzte Speicher bezeichnet, in dem alle temporären Wert von Variablen und Puffern während der Laufzeit einer Applikation abgelegt werden.

Beachten Sie, daß CA-Clipper Sommer '87 den Speicher 'bei Bedarf' vom verfügbaren DOS-Speicher belegt. Solange nicht MEMORY(0), BLIMEMPAK(-1) oder BLIMEMBLK() ausgeführt werden (was CA-Clipper dazu veranlaßt allen freien DOS-Speicher dem FREE POOL hinzuzufügen), zeigt diese Funktion nur den aktuell belegten FREE POOL an.

Beispiel:
```
*  Ermittle gesamten, derzeit benutzten
* dynamischen Speicher:
*
? BLIMEMUSE()," Bytes werden derzeit benutzt. "
```

Siehe auch: BLIMEMAVL()
BLIMEMSIZ()

BLIMGRSTS()

Funktion: Programmstatusinformationen ermitteln

Syntax: `nlWert = BLIMGRSTS(niOption)`

Parameter: niOption : Optionsnummer für Statusinformation

Rückgabe: unterschiedlich

Bemerkung: BLIMGRSTS() ist eine Mehrzweckfunktion, die dazu verwendet werden kann, Statusinformationen über das derzeit ausgeführte Programm (samt Umgebung) zu erlangen.

Diese Funktion kann aus den meisten Sprachen heraus über die normalen Aufrufkonventionen der jeweiligen Sprache aufgerufen werden. BLIMGRSTS() nimmt einen Parameter entgegen - eine Optionsnummer, die die angeforderte Statusinformation auswählt. Solche Informationen, die in der aktuellen Umgebung gerade nicht definiert sind, werden als Null zurückgegeben (z.B. die Cache-Größe in einem Protected Mode-Programm).

Die Definitionen zu den konstanten Parametern von BLIMGRSTS() finden Sie in den Header-Dateien BLINKER.*. Derzeit sind folgende Werte definiert:

BliHostMode DOS-Extender-Host-Modus (DPMI / VCPI / XMS)

BliMachineMode aktueller Rechnermodus (Real, Protected)

BliCacheLoc Bereich, in dem sich der Realmode Overlay- Cache befindet (EMS / XMS)

BliCacheSize Größe des Overlay-Cache

BliOverlayLoc Speichertyp des Overlay-Bereichs

BliOverlaySize Größe des Overlay-Bereichs

BliExtMemAvail verfügbarer Erweiterungsspeicher

BliRealMemAvail verfügbarer konventioneller Speicher

BliVirMemAvail verfügbarer virtueller Speicher

Beispiel:

```
*  Prozessor- und Overlay-Cache-Status ausgeben
*
if (BLIMGRSTS(BliMachineMode) = BliModeReal)
  ? "Arbeitsmodus ist der Real Mode"
  cacheloc = BLIMGRSTS(BliCacheLoc)
  do case
  case cacheloc = BliCacheNone
     ? "Es steht kein Cache zur Verfügung"
  case cacheloc = BliCacheXMS
     ? "Der Cache befindet sich im XMS"
  case cacheloc = BliCacheEMS
     ? "Der Cache befindet sich im EMS"
  endcase
else
  ? "Arbeitsmodus ist der Protected Mode"
  hosttype = BLIMGRSTS(BliHostMode)
  ? "Dos-Extender-Host ist : "
  do case
  case hosttype = BliHostDPMI
     ?? "DPMI"
  case hosttype = BliHostVCPI
     ?? "VCPI"
  case hosttype = BliHostXMS
     ?? "XMS"
  endcase
endif
```

BLIOVLCLR()

Funktion: Löschen von nicht aktiven CA-Clipper Sommer '87 Overlays aus dem Speicher

Syntax: `BLIOVLCLR()`

Parameter: Keine

Rückgabe: Keine

Bemerkung: Diese Funktion ist nur dann aktiv, wenn der Befehl BLINKER OVERLAY DYNAMIC als Speicherbelegungs-Methode angegeben wurde (nur CA-Clipper Sommer ’87). Sie hat keine Funktion, wenn BLINKER OVERLAY FIXED angegeben wurde.

Diese Funktion veranlaßt den BLINKER-Overlay-Manager, alle inaktiven Overlay-Prozeduren aus dem Overlay-Pool zu löschen oder freizugeben, und gibt den von diesen Prozeduren bisher benutzten Speicher an CA-Clipper zurück.

Diese Funktion kann im Hauptmodul einer Applikation eingesetzt werden, um sicherzustellen, daß CA-Clipper das Maximum an dynamischen Speicher zur Verfügung hat, bevor ein neues Overlay ausgeführt wird.

Wenn alle Overlays aus dem Speicher entfernt worden sind, wird die Arbeitsgröße des Overlay-Pools auf das Maximum von der zu Programmstart vorgegebenen Größe und der

Gesamtgröße alle aktuell aktiven Overlays eingestellt. Die zu Programmbeginn vorgegebene Größe wird durch die letzte Verwendung des Befehls BLINKER OVERLAY OPSIZE <niGröße> oder des Parameters /OOnn in der Umgebungsvariablen BLINKER bestimmt.

Beispiel:
```
*  Löschung alle inaktiven Overlays
*
BLIOVLCLR()
```

Siehe auch: BLINKER OVERLAY DYNAMIC
BLINKER OVERLAY OPSIZE
BLIMEMPAK()

BLIOVLOPS()

Funktion: Arbeitsgröße des Overlay-Pools ermitteln

Syntax:
```
nlWert = BLIOVLOPS()
```

Parameter: Keine

Rückgabe: Arbeitsgröße des Overlay-Pools in Bytes

Bemerkung: Diese Funktion gibt die Größe des Overlay-Pools in Bytes zurück.

Die Arbeitsgröße kann zur Linkezeit mit dem Befehl BLINKER OVERLAY OPSIZE <nlGröße> oder zur Laufzeit mit dem Parameter /OOnn in der Umgebungsvariablen BLINKER festgelegt werden (siehe Kapitel 9, BLINKER OVERLAY OPSIZE).

Der Wert, der durch diese Funktion zurückgegebn wird, repräsentiert die verlangte Größe des Overlay-Bereichs zu der Zeit, zu der das Programm startet. Dieser Wert wird nahe dem durch den während der Linkzeit benutzten Beflehl BLINKER OVERLAY OPSIZE nnn und der /OOnnn Einstellung der Blinker Umgebungsvariablen sein. Die tatsächliche Größe des belegten Bereiches mag variieren in Abhängigkeit von der Verfügbarkeit von EMS und UMB's und der Einstellung der zugehörigen BLINKER Optionen. Sehen Sie bitte auch unter BLIOVLSIZ(), um nähere Informationen zur tatsächlich belegten Größe zu erfahren.

Hinweis zu CA-Clipper Sommer '87 in Verbindung mit der DYNAMIC Methode:

Wird BLINKER OVERLAY DYNAMIC verwendet (die Standard-Einstellung) um Overlay-Speicher zu belegen, ist dies die maximale Menge an Speicher, den Blinker für die Belegung von Speicher für Overlays belegen wird. Hat Blinker einmal die gegebenen Menge Speicher verwendet, werden unbenutzte Overlays aus dem Speicher entfernt, um Platz für weitere Overlays zu schaffen, anstatt weiterhin Speicher von CA-Clipper anzufordern, um dort die Overlays zu plazieren.

Während der Ausführung kann der Overlay-Manager gezwungen sein, diesen Wert zu erhöhen, falls Overlays stark verschachtelt wurden, da sich bei der DYNAMIC-Methode alle aktiven Overlays im Speicher befinden müssen. Dieser Wert bleibt dann solange gültig, is das Programm beendet wurde, oder BLIOVLCLR() ausgeführt wurde. In diesem Falle wird der Wert für die OPSIZE wieder auf Ihren ursprünglichen Wert zurückgesetzt.

Beispiel:
```
*  Ermittlung der Arbeitsgröße des Overlay-Pools:
*
? "Die Arbeitsgröße des Overlay-Pools beträgt "
?? BLIOVLOPS(), "Bytes"
```

Siehe auch: BLINKER OVERLAY OPSIZE

BLIOVLRES()

Funktion: Wiederaufnahme der Arbeit des Overlay-Managers

Syntax:
```
BLIOVLRES()
```

Parameter: Keine

Rückgabe: Keine

Bemerkung: Diese Funktion sorgt für die Wiederaufnahme der Arbeit des Overlay-Managers, nachdem dieser mittels BLIOVLSUS() angehalten wurde. Der Overlay-Manager wird sich selbst re-initialisieren, die Overlay-Datei(en) öffnen und mit seiner Arbeit fortfahren.

Siehe auch: BLIOVLSUS()

BLIOVLSIZ()

Funktion: Aktuelle Größe des Overlay-Pools

Syntax:
```
nlWert = BLIOVLSIZ()
```

Parameter: Keine

Rückgabe: Aktuelle Größe des Overlay-Pools

Bemerkung: Diese Funktion gibt die tatsächliche Größe des Overlay-Pool beimStart des Programmes in Bytes zurück. Die Standardgröße ist der zuletzt entweder zur Linkzeit mit dem Blinkerbefehl BLINKER OVERLAY OPSIZE nnn oder zur Laufzeit mit der Einstellung /Oonnn in der Umgebungsvariablen BLINKER gesetzte Wert. Diese Größe kann auf zei Arten verändert werden:

Wurde der BLINKER OVERLAY PAGEFRAME ON-Befehl zur Linkzeit eingesetzt, oder der /OP1 Schalter in der BLINKER-Umgebungsvariablen zur Laufzeit gesetzt, dann wird, sofern ein EMS Pageframe vorhanden ist und die Standard-OPSIZE kleiner als 64KByte ist, die Größe des Overlay-Bereichs um so viel erweitert, daß sie die Pageframe voll ausnutzt. Diese neue Größe wird dann die Größe sein, die durch BLIOVLSIZ() zurückgegeben wird.

Ist die Benutzung des Pageframe nicht möglich oder ausgeschaltet, und voraugesetzt, die Benutzung von UMB's ist nicht gesperrt durch Angabe des Befehls BLINKER OVERLAY UMB OFF zur Linkzeit oder Angabe von /OU0 in der BLINKER-Umgebungsvariablen zur Laufzeit, dann wird Blinker versuchen, mindestens soviel UMB zu belegen, wie durch die Standardgröße der OPSIZE vorgegeben ist. Ist dies erfolgreich, versucht Blinker soviel UMB-Speicher für Overlay-Operationen zu bekommen wie möglich. Diese neue Größe

wird dann als Rückgabewert von BLIOVLSIZ() zurückgeliefert.

Ist keine der beiden Methoden möglich, oder sind beide Methoden abgeschaltet, dann wird soviel Speicher belegt, wie der Standardwert von OPSIZE angibt.

Hinweis zu CA-Clipper Sommer '87 in Verbindung mit der DYNAMIC Methode:

Wird die BLINKER OVERLAY DYNAMIC Methode zur Speicherbelegung für Overlays gewählt, repräsentiert dieser Wert die exakte Menge Speicher, die Blinker zum derzeitigen Zeitpunkt für Overlays benötigt. Zu Beginn einer Applikation ist die aktuelle Größe des Overlay-Pools Null. Sie erhöht sich nach und nach, bis die mit BLINKER OVERLAY OPSIZE <nGröße> eingestellte und von BLIOVLOPS() zurückgegebene Arbeitsgröße (=Maximalgröße) des Overlay-Pools erreicht wird,

Wird die Arbeitsgröße erreicht, so bleibt die aktuelle Größe des Overlay-Pools so lange an diesem Punkt, bis die Funktion BLIOVLCLR() aufgerufen wird. Wenn ein Aufruf von BLIOVLCLR() erfolgt ist, werden alle inaktiven Overlays entfernt und der freigewordene Speicher an CA-Clipper zurückgegeben. Die aktuelle Größe des Overlay-Pools wird somit auf die Größe der aktiven Overlays reduziert. Im besten Fall ergibt der Rückgabewert von BLIOVLSIZ() nach Aufruf von BLIOVLCLR() im Rootbereich NULL.

Beispiel:
```
* Ermittle die aktuelle Größe des Overlay-Pools:
*
? "Größe des Overlay-Pools",BLIOVLSIZ()," Bytes"
```

Siehe auch: BLINKER OVERLAY OPSIZE

BLIOVLSUS()

Funktion: Beenden des Overlay-Managers

Syntax: `BLIOVLSUS(niWert)`

Parameter: niWert : Grad der Abschaltung

Rückgabe: Keine

Bemerkung: Diese Funktion beendet den Overlay-Manager. Abhängig vom Grad der Abschaltung gibt der Overlay-Manager einige oder alle seiner beanspruchten System-Resourcen wieder frei. Die numerischen Werte von 1 bis 3 sind gültige Parameter und die freigegebenen System-Resourcen für jeden Parameter schlüsseln sich wie folgt auf:

- 1 Schließt alle offenen Overlay-Dateien, DOS-Filehandles werden wieder freigegeben.
- 2 wie (1), jedoch wird auch der EMS/XMS-Speicher, den der Overlay-Manager belegte, wieder freigegeben.
- 3 ALLE Resourcen werden freigegeben, einschließlich des Overlay-Pools.

Wird der Overlay-Manager mittels BLIOVLRES() wieder gestartet, so versucht dieser, alle schon mal in Anspruch genommenen Resourcen wieder zu belegen. Kann er dies nicht, so wird das Programm mit einer BLINKER-Fehlermeldung Nr. 1216, 'Der Overlay-Manager ist nicht in der Lage fortzufahren' beendet. Wird der Overlay-Bereich erfolgreich

wiederbelegt, aber es gibt keine freien Resourcen für den Overlay-Puffer, wird die Ausführung der Applikation fortgeführt, ohne die Routinen im Overlay zu puffern.

Anmerkung: Diese Routine muß vom Root des Programms aus aufgerufen werden, da der Overlay-Manager inaktiv ist, sobald die Funktion BLIOVLSUS() durchgeführt wurde und erst wieder aktiv werden kann, nachdem ein BLIOVLRES() gesetzt werden konnte.

Siehe auch: BLIOVLRES()

BLIPTRDEC()

Funktion: Huge Zeiger dekrementieren

Syntax: `LPVOID BLIPTRDEC(lpPointer, niWert)`

Parameter: lpPointer : Zeiger auf Speicher, der dekrementiert wird

niWert : Wert, um den der Zeiger dekrementiert wird

Rückgabe: Dekrementierter Zeiger

Bemerkung: Die Funktion BLIPTRDEC() (decrement huge pointer) ist die Umkehrfunktion zu BLIPTRINC() - mehr Information zu dieser Funktion finden Sie unter BLIPTRINC().

Beispiel: `Wenn BLOCK z.B. irgend ein komplexer Typ ist, dann :`

```
1 memptr = (BLOCK huge *) halloc ((long) BLKMAX,sizeof (BLOCK));
2 if (memptr)
3     {
4     tmpptr = memptr;
5     for (i = 0; i < BLKMAX; i++)
6         {
7         ...              /* irgendwelche Operationen */
8         tmpptr--;        /* auf den vorherigen Block zeigen */
9         }
10    }
```

`Zeile 8 kann ersetzt werden mit:`

```
tmpptr = BLIPTRDEC (tmpptr, (long) sizeof (BLOCK));
```

Siehe auch: BLIPTRINC()

BLIPTRINC()

Funktion: Huge Zeiger inkrementieren

Syntax: `LPVOID BLIPTRINC(lpPointer, niWert)`

Parameter: lpPointer : Zeiger auf Speicher, der inkrementiert wird

niWert : Wert, um den der Zeiger inkrementiert wird

Rückgabe: Inkrementierter Zeiger

Bemerkung: Einige C bzw C++ Compiler generieren Code, in dem Huge Protected-ModeZeiger einwandfrei funktionieren (wie zum Beispiel Borland C++ in den meisten Fällen), während andere Referenzen auf __AHSHIFT und __AHINCR generieren. Diese Symbole werden dazu verwendet, Huge Zeiger über Segmentgrenzen hinweg zu 'bewegen'. Standardmäßig belegen die BLXxxxnn-Bibliotheken diese Symbole mit ihren Protected-Mode-Werten, was bei einem Dual Mode Programm, das im Real Mode läuft, ein Problem hervorrufen kann. Es gibt zwei Lösungen dazu:

BLXSHFTR.OBJ oder BLXSHFTP.OBJ können als FILE hinzugelinkt werden, um diese Symbole mit Protected Mode oder Real Mode kompatiblen Werten aufzulösen. Aber die resultierende EXE kann nur im ausgewählten Modus fehlerfrei laufen, weil die Inkremente für den jeweiligenModus fest kodiert sind.

Der Quelltext kann so geändert werden, daß die zwei Blinker-Funktionen BLIPTRINC() und BLIPTRDEC() verwendet werden, um die Zeiger in allen Modi korrekt zu 'bewegen', einschließlich DUAL Mode

Beispiel: Sei z.B. BLOCK irgendein komplexer Typ, dann :

```
1 memptr = (BLOCK huge *) halloc ((long) BLKMAX,sizeof (BLOCK));
2 if (memptr)
3     {
4     tmpptr = memptr;
5     for (i = 0; i < BLKMAX; i++)
6         {
7         ...           /* Irgendwelche Operationen */
8         tmpptr++;     /* auf nächsten Block zeigen */
9         }
10    }
```

Zeile 8 kann wie folgt ersetzt werden :

```
tmpptr = BLIPTRINC (tmpptr, (long) sizeof (BLOCK));
```

Siehe auch: __BLIPTRDEC()

BLISERNUM / BLISERNUMBAS()

Funktion: Interne Seriennummer auslesen

Syntax: `cWert = BLISERNUM()`

Parameter: Keine

Rückgabe: Seriennummer in String Form

Bemerkung: Diese Funktion untersucht eine Seriennummer und andere Informationen von insgesamt bis zu 100 Zeichen Länge, die zur Linkzeit verschlüsselt in die .EXE-Datei eincodiert worden sind, um z.B. eine Seriennummer als Kopierschutz zu verstecken.

Die Informationen werden als String von maximal 100 Zeichen Länge in der .EXE-Datei codiert gespeichert und zur Laufzeit wird der String exakt in demselben Format zurückgegeben.

Dies Funktion steht zur Zeit nur für DOS und DOS Extender Programmen zur Verfügung, nicht für Windowsprogramme.

Beispiel:

```
Wenn Sie z.B. in eine Applikation die Seriennummer 00100001, eine Lizenz für
5 Benutzer und den Namen des Lizenznehmers (im Beispiel BLINK)
hineincodieren möchten, würden Sie folgenden Befehl verwenden:

BLINKER EXECUTABLE SERIAL 00100001 5 Blink, Inc.

Während der Laufzeit der Applikation können diese Werte dann eingelesen
werden, in dem der Rückgabestring von BLISERNUM() mit SUBSTR() in
entsprechende Teilstings aufgesplittet wird:

TMPSERNUM = BLISERNUM()
? "SERIENNUMMER.....:",Substr(TMPSERNUM,1,8)
? "Anzahl Benutzer..:",Val(Substr(TMPSERNUM,10,1))
? "Lizenznehmer.....:",Substr(TMPSERNUM,12,20)
```

Siehe auch: BLINKER EXECUTABLE SERIAL

BLISTRFRG()

Funktion: String mit Fragmentierungsinformationen für CA-Clipper Sommer ’87 liefern

Syntax: `cWert = BLISTRFRG([cString])`

Parameter: cString : String mit Blockzeichen

Rückgabe: Fragmentierung als String

Bemerkung: Diese Funktion gibt einen String mit einem Überblick über den dynamischen Speicher zurück, um das Ausmaß der Fragmentierung aufzuzeigen. Jedes Zeichen repräsentiert bis zu 512 Bytes. Aufeinanderfolgende gleiche Zeichen stellen demnach zusammenhängende Blöcke von mehr als 512 Bytes dar.

Optional kann ein String aus fünf Zeichen in Form "ffuua" als Parameter übergeben werden(siehe Beschreibung unten). Die ersten beiden Zeichen (ff) stellen die freien Blöcke dar und die nächsten beiden Zeichen (uu) die benutzten Blöcke. Das abschließende Zeichen (a) wird zur Darstellung des nächsten Speicherbelegungspunktes verwendet. Der Rückgabestring enthält eine Liste des Near Heap und des Far Heap (FREE POOL), voneinander durch ein ‘,’-Zeichen getrennt. Layout und Bedeutung der Anzeige sind in der Erläuterung zu der Funktion BLIDISFRG() genau erklärt.

Beispiel:

```
* String mit der Anzeige der Speicherfragmentierung ausgeben
* ‘f’ für alle freien und ‘u’ für alle benutzten Blöcke

TMPSTRFRG = BLISTRFRG("ffuua")
? "Der Zustand des dynamischen Speichers ist derzeit"
? TMPSTRFRG
```

Siehe auch: BLILSTFRG()
BLIDISFRG()

BLIVERNUM()

Funktion: Gibt die Versionsnummer des Blinker als Integerwert zurück

Syntax:
```
nuWert := BLIVERNUM()
```

Parameter: keine

Rückgabe: Nummer der Blinker-Version, die zum Linken des Programms verwendet wurde.

Bemerkung: Diese Funktion gibt einen Integerwert zurück, der die Versionsnummer des Blinker enthält, mit der das Programm gelinkt wurde. Diese stellt die Hauptversion * 100 + Nebenversion dar. Zum Beispiel repräsentiert 300 die Version 3.00.

Beispiel:
```
* BLIVERNUM()
*
? "Die Blinker-Versionsnummer lautet"
?? STR (BLIVERNUM()/100, 2)
```

DosAllocHuge()

Funktion: Belegt einen Block von bis zu 16 MB im Erweiterungsspeicher.

Syntax:
```
USHORT BLXAPI DosAllocHuge(USHORT segcount,
                           USHORT lastsize,
                           PSEL selector,
                           USHORT segmax,
                           USHORT flags);
```

Parameter:

segcount	Anzahl der vollständigen 64-KB-Segmente.
lastsize	Bytes im letzten Segment (0=keins).
selector	Zeiger auf eine Integervariable, die den anfänglichen Selektor des Blocks aufnimmt.
segmax	Maximale Anzahl von 64-KB-Blöcken, die später für den Block reserviert werden mit Aufruf von DosReallocHuge (). (Kann auch 0 sein, siehe unten).
flags	reserviert, muß 0 sein

Rückgabe: Der Basisselektor des Huge-Blocks.

Bemerkung: Diese Funktion belegt einen Speicherblock von bis zu 16 MB und gibt einen Selektor zurück, der auf die Basis des Blocks zeigt. Ist der Block größer als ein 64-KB-Segment, dann wird ein zusammenhängender Bereich von Selektoren reserviert, und jedem vollen oder teilweise gefülltem Segment wird ein Selektor zugewiesen. Das Inkrement zum Bewegen von einem Selektor zum nächsten kann über DosGetHugeShift() berechnet werden.

Segmax gibt die maximale Anzahl der 64-KB-Blöcke an, auf die der Block vergrößert werden kann. Über DosAllocHuge werden genügend nacheinanderfolgende Selektoren belegt, um den Erfolg nachfolgender Neubelegungen abzusichern. Ein Wert von 0 für segmax verursacht, daß DosAllocHuge() einen segmax-Wert gleich der angeforderten

Anfangsbelegungsgröße belegt, um so zu verhindern, daß eine erneute Belegung mehr als (segcount+1) 64-KB-Segmente verwendet.

Beispiel:

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <blx286.h>

void main (void)
{
SEL      selbase;
USHORT   hshift;
USHORT   ret;
int      selinc;
char  *fp;

/* Alloziiert einen 1-MB-Block */
if ((ret = DosAllocHuge(16,0,&selector,0,0)) != 0)
  {
  printf("DosAllocHuge() fehlgeschlagen");
  exit(255);
  }
DosGetHugeShift(&hshift);
selinc = 1 << hshift;
/* alle 64-KB-Blöcke mit NULLEN füllen */
tmpsel = selector;
for (cnt = 0; cnt <16; cnt++, tmpsel += selinc);
  {
  fp = MK_FP(tmpsel,0);
  *fp = 0;      /* Fill first character */
  fp++;         /* Advance to next character */
  memset(fp, 0, 0xFFFFh);
  printf("gelöschtes Segment : %04X\n",tmpsel);
  }
/* Block freigeben */
DosFreeSeg(selector);
}
```

Siehe auch: DosFreeSeg()

DosAllocRealSeg()

Funktion: Belegt Real Mode-Speicherblock.

Syntax:

```
USHORT BLXAPI DosAllocRealSeg(ULONG size,
                              PUSHORT paragraph,
                              PSEL selector);
```

Parameter: size Größe des zu belegenden Blocks in Byte. (Kann nicht größer als 64 KB sein).

paragraph	Zeiger auf einen Integerwert, um die Real Mode- Segmentadresse des Speicherblocks aufzunehmen.
selector	Zeiger auf auf einen Integerwert, um den Protected Mode-Selektor des Speicherblocks aufzunehmen.

Rückgabe: Die Real Mode-Abschnittsadresse und der Protected Mode-Selektor des konventionellen Speicherblocks.

Bemerkung: Weil diese Funktion sowohl die Adresse eines Real Mode-Paragraphen als auch einen Protected Mode-Zeiger zurückgibt, kann der belegte Block in beiden Modi adressiert werden, was für die Mixed-Mode-Programmierung nützlich sein kann.

Beachten Sie, daß ein durch DosFreeRealSeg() freigegebener Speicher nur an den DOS Extender zurückgegeben wird und nicht an DOS, bevor das Programm beendet wird. Daher kann DosRealAvail() die Zunahme an Speicher nicht erkennen, obwohl DosAllocRealSeg() den Speicher wieder neu belegen kann.

Beispiel:

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <blx286.h>

void main (void)
{
SEL      selector;
USHORT   paragraph;

/* Einen 16-KB-Block belegen */
if (DosAllocRealSeg((16L << 10), &paragraph, &selector) != 0)
  {
  printf("DosAllocRealSeg fehlgeschlagen");
  exit(255);
  }
printf("Real-Segment: %04X Prot-Selektor: %04X\n",
    paragraph,selector);
/* den Block wieder freigeben */
DosFreeRealSeg(selector);
}
```

Siehe auch: DosFreeRealSeg()
DosRealAvail()

DosAllocSeg()

Funktion: Belegt einen Block von bis zu 64 KB im Erweiterungsspeicher.

Syntax:
```
USHORT BLXAPI DosAllocSeg(USHORT size,
                          PSEL selector,
                          USHORT flags);
```

Parameter:

size	Blockgröße in Bytes (0=65536).
selector	Zeiger auf Integer zum Ablegen des Selektors.
flags	Reserviert (muß 0 sein).

Rückgabe: Selektor für den Erweiterungsspeicherblock.

Bemerkung: DosAllocSeg() belegt ein Speicherblock mit einer maximalen Größe von 65536 Bytes (64 KB).

Die meisten High-Level-Sprachen haben eigene Funktionen zur Reservierung von Speicher, wie z.B. malloc() in C. Diese Funktionen können statt DosAllocSeg() für Speicherblöcke < 64KB verwendet werden. Sie eignen sich besonders, wenn die Unterschiede im Quellcode zwischen Real Mode und Protected Mode minimal sein sollen.

Die Größe des verfügbaren Speichers zur Verwendung von DosAllocSeg() läßt sich mit der Funktion DosMemAvail() ermitteln .

Beispiel:
```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <blx286.h>

void main (void)
{
SEL      selector;

/* Einen 4-KB-Block belegen */
if (DosAllocSeg(4096, &selector, 0) != 0)
  {
  printf("DosAllocSeg() fehlgeschlagen \n");
  exit(255);
  }
printf("4-KB-Block belegt bei: %04X\n",selector);
/* Block wieder freigeben */
DosFreeSeg(selector);
}
```

Siehe auch: DosFreeSeg()

DosCreateCSAlias()

Funktion: Erzeugt einen ausführbaren (Code) Selektor, der einem Datenselektor entspricht.

Syntax:
```
USHORT BLXAPI DosCreateCSAlias( SEL datasel,
                                PSEL codesel);
```

Parameter:

datasel	Datenselektor, der einen Alias erhalten soll. Dieser Selector muß zu einem einzelnen Datensegment mit einer Größe <= 64KB gehören.
codesel	Zeiger auf eine Variable, in der der zurück-gegebene Alias-Codeselektor abgelegt werden soll.

Rückgabe: Ausführbarer Selektor, der einem Daten-Selektor entspricht.Dieser Selektor hat die gleiche lineare Basisadresse und Länge wie der ursprüngliche Daten-Selektor, aber mit dem Attribut READ/ EXECUTE CODE.

Bemerkung: Diese Funktion ermöglicht, daß in einem Datensegment enthaltener Code ausgeführt werden kann.

Hinweis: Der ursprüngliche Data-Selektor muß im Speicher gesperrt werden (mit DosLockSeg()) bevor ein Alias-Selektor erzeugt wird. Damit wird sichergestellt, daß der ursprüngliche Selektor nicht aus dem Speicher geswappt und später mit einer neuen linearen Basisadresse von der Platte zurückgeladen werden kann. Dies ist notwendig, weil ein Alias-Selektor die Änderungen des ursprünglichen Selektors nicht mitverfolgen kann. Die Sperrung des originalen Data-Selektors kann mit DosUnlockSeg() wieder aufgehoben werden, nachdem der Alias-Selektor mit DosFreeSelector() freigegeben worden ist.

Beispiel:
```
#include <stdio.h>
#include <blx286.h>

#define TOSCREEN   2
#define TOPRINT 5

unsigned char codebyte[] =
{
  0xb8, 0x00, 0x00,             /* mov AX, 0 */
  0xba, 0x00, 0x00,             /* mov DX, 0 */
  0xcd, 0x21,                   /* int 21    */
  0xcb                       /* retf      */
};

void (far *execdata) (void);

void output(char *string, unsigned char device)
{
while (*string)
  {
  codebyte[2] = device;      /* Bildschirm/Drucker */
  codebyte[4] = *string;     /* Zeichen zur Ausgabe */
  (*execdata)();
  string++;
  }
```

```
}

void main (void)
{
/* Das ist ein konstruiertes Beispiel */
void far    *fp;
SEL      codesel;
/* Segment zunächst sperren */
fp = (void far *)(codebyte);
DosLockSeg(SELECTOROF(fp));
if (DosCreateCSAlias(SELECTOROF(fp),&codesel) == 0)
  {
  execdata = MK_FP(codesel, OFFSETOF(fp));
  output("Ein konstruiertes Beispiel\n",TOSCREEN);
  output("Ein konstruiertes Beispiel\n",TOPRINT);
  DosFreeSelector(codesel);
  }
}
```

Siehe auch: DosCreateDSAlias()
DosLockSeg()

DosCreateDSAlias()

Funktion: Erzeugt einen Datenselektor, der einem ausführbaren (Code) Selektor entspricht.

Syntax:
```
USHORT BLXAPI DosCreateDSAlias( SEL codesel,
                                PSEL datasel);
```

Parameter:

codesel	Codeselektor, der einen Alias erhalten soll.
datasel	Zeiger auf eine Variable, in der der zurück-gegebene Alias-Datenselektor abgelegt werden soll

Rückgabe: Datenselektor, der einem ausführbaren Code-Selektor entspricht. Dieser Selector hat die gleiche lineare Basisadresse und Länge wie der ursprüngliche Code-Selektor, aber mit dem Attribut READ/ WRITE CODE.

Bemerkung: Diese Funktion ermöglicht, daß ein Codesegment beschrieben und aktualisiert werden darf.

Hinweis: Der ursprüngliche Code-Selektor muß im Speicher gesperrt werden (mit DosLockSeg()) bevor ein Alias-Selektor erzeugt wird. Damit wird sichergestellt, daß der ursprüngliche Selektor nicht aus dem Speicher geswappt und später mit einer neuen linearen Basisadresse von der Platte zurückgeladen werden kann. Dies ist notwendig, weil ein Alias-Selektor die Änderungen des ursprünglichen Selektors nicht mitverfolgen kann. Die Sperrung des originalen Code-Selektors kann mit DosUnlockSeg() wieder aufgehoben werden, nachdem der Alias-Selektor mit DosFreeSelector() freigegeben worden ist.

Beispiel:
```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <blx286.h>
```

```
void main (void)
{
SEL   datasel;

/* Segment zunächst sperren */
DosLockSeg(SELECTOROF(test));
if (DosCreateDSAlias (SELECTOROF (test), &datasel) != 0)
  {
  printf ("DosCreateDSAlias fehlgeschlagen\n");
  exit (255);
  }
printf ("Code-Selektor für Test() ist  %04X\n",
            SELECTOROF (test));
printf ("   Data-Selektor ist   %04X\n\n",datasel);
DosFreeSelector(datasel);
}
```

Siehe auch: DosCreateCSAlias()
DosLockSeg()

DosFreeRealSeg()

Funktion: Gibt einen Selektor frei, der zuvor über DosAllocRealSeg() belegt wurde.

Syntax: `USHORT BLXAPI DosFreeSeg(SEL selector);`

Parameter:

selector	Protected Mode-Selektor eines konventionellen Speicherblocks, der über DosAllocRealSeg() belegt wurde.

Rückgabe: Keine

Bemerkung: Mit dieser Funktion werden konventionelle Speicherblöcke, die mit DosAllocRealSeg() belegt wurden, wieder freigegeben. Sie sollte nicht zur Freigabe anderer Arten von Selektoren oder Speicherblöcke verwendet werden.

Beachten Sie, daß ein durch DosFreeRealSeg() freigegebener Speicher nur an den DOS Extender zurückgegeben wird und nicht an DOS, bevor das Programm beendet wird. Daher kann DosRealAvail() die Zunahme an Speicher nicht erkennen, obwohl DosAllocRealSeg() den Speicher wieder neu belegen kann.

Beispiel:

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <blx286.h>

void main (void)
{
SEL         selector;
USHORT      paragraph;

/* Real Mode-4KB-Block belegen */
if (DosAllocRealSeg(4096L, &paragraph, &selector) != 0)
  {
  printf("DosAllocRealSeg fehlgeschlagen\n");
  exit(255);
  }
printf("Real Mode-4K-Block angelegt \n");
```

```
printf("Selektor des Blocks : %04X\n",selector);
printf("Segment  des Blocks : %04X\n",paragraph);
if (DosFreeRealSeg(selector) != 0)
  {
  printf("DosFreeRealSeg()-Fehler - Selektor = %04X\n",selctor);
  exit(254);
  }
}
```

Siehe auch: DosAllocRealSeg()
DosFreeSeg()
DosFreeSelector()

DosFreeSeg()

Funktion: Gibt einen Speicherblock frei, der zuvor über DosAllocSeg(), DosAllocHuge() belegt wurde.

Syntax:
```
USHORT BLXAPI DosFreeSeg(SEL selector);
```

Parameter: selector — Protected Mode-Selektor eines mit DosAllocSeg() oder DosAllocHuge() angelegten Speicherblocks.

Rückgabe: Keine

Bemerkung: Mit dieser Funktion können Speicherblöcke, die mit DosAllocSeg() oder DosAllocHuge() angelegt wurden, freigegeben werden. Sie sollte nicht für andere Selektortypen verwendet werden.

Beispiel:
```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <blx286.h>

void main (void)
{
SEL   selector;

/* 4KB-Block belegen */
if (DosAllocSeg(4096, &selector, 0) != 0)
  {
  printf("DosAllocSeg fehlgeschlagen\n");
  exit(255);
  }
printf("4K-Block angelegt bei: %04X\n",selector);
if (DosFreeSeg(selector) != 0)
  {
  printf("DosFreeSeg()-Fehler - Selektor = %04X\n",selector);
  exit(254);
  }
}
```

Siehe auch: DosFreeRealSeg()
DosFreeSelector()

DosFreeSelector()

Funktion: Gibt einen Selektor frei, der zuvor über DosMapRealSeg(), DosCreateCSAlias() oder DosCreateDSAlias() angelegt wurde.

Syntax:
```
USHORT BLXAPI DosFreeSelector(SEL selector);
```

Parameter: selector — Selektor, der über DosMapRealSeg(), DosCreateCSAlias() oder DosCreateDSAlias() angelegt wurde.

Rückgabe: Keine

Bemerkung: Mit dieser Funktion werden Alias-Selektoren (angelegt mit DosCreateDSAlias() oder DosCreateCSAlias()) oder mit DosMapRealSeg() erzeugte Selektoren freigegeben. Sie sollte nicht für andere Selektortypen verwendet werden.

Beispiel:
```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <blx286.h>

#define VID_BUF 0xB800
#define VID_SIZE 0x8000l

unsigned char far *VideoBuffer;

void main (void)
{
SEL      selector;
/* Hole Selektor, der dem Bildschirmpuffer entspricht */
if (DosMapRealSeg(VID_BUF,VID_SIZE,&selector) != 0)
  {
  printf("DosMapRealSeg fehlgeschlagen\n");
  exit(255);
  }
VideoBuffer = MK_FP(selector,0);
*VideoBuffer[0] = "X";       /* Aus den Bildschirm schreiben */
if (DosFreeSelector(selector) == 0)
  printf("Selektor für Video-Puffer ist nun ungültig\n");
}
```

Siehe auch: DosFreeRealSeg(), DosFreeSeg()

DosGetBIOSSeg()

Funktion: Gibt einen Selektor für den 1-KB-BIOS-Datenbereich zurück.

Syntax: `USHORT BLXAPI DosGetBIOSSeg(PSEL biossel);`

Parameter:

biossel	Zeiger auf eine Variable für den zurückge-gebenen Protected Mode-Selektor für den BIOS-Datenbereich.

Rückgabe: Selektor für den BIOS-Datenbereich, entsprechend dem Real Mode Segment 0x0040.

Bemerkung: Far-Zeiger auf den BIOS-Datenbereich müssen über den durch diese Funktion zurückgegebenen Selektor gebildet werden. Der von dieser Funktion zurückgegebene Selektor darf NIEMALS freigegeben werden.

Beispiel:

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <blx286.h>

void main (void)
{
SEL      biossel;
/* Gibt einen Selektor für den BIOS-Datenbereich zurück */
if (DosGetBIOSSeg(&biossel) != 0)
      {
      printf("DosGetBIOSSeg() fehlgeschlagen\n");
      exit(255);
      }
  printf("BIOS-Selektor ist : %04X\n",biossel);
}
```

DosGetExceptionHandler()

Funktion: Gibt eine Protected Mode-Adresse auf einen Ausnahme-Handler zurück

Syntax:

```
USHORT BLXAPI DosGetExceptionHandler( USHORT exception,
                                      PEXCHAN __far *handler);
```

Parameter:

exception	Nummer der Ausnahme (0-13)
handler	Zeiger auf einen Funktionszeiger für die zurückgegebene Adresse des Ausnahme-Handlers.

Rückgabe: Adresse des Ausnahme-Handlers.

Bemerkung: DosGetExceptionHandler() gibt die Adresse des aktuellen Handlers für die angegebene Ausnahmenummer zurück.

Beispiel:

```
#include <stdio.h>
#include <blx286.h>

void main (void)
{
```

```
EXCHAN handler;
if (DosGetExceptionHandler(0x0D, &handler) == 0)
  printf("Handler für Exception 13: %Fp\n",handler);
}
```

Siehe auch: DosSetExceptionHandler()

DosGetHostMode()

Funktion: Gibt den aktuellen DOS-Extender-Host-Modus zurück (DPMI, VCPI, XMS).

Syntax: `USHORT BLXAPI DosGetHostMode(BYTE __far *hostmode);`

Parameter: hostmode Zeiger auf eine Byte-Variable für den Hostmode

Rückgabe: Anzeigebyte für den Host-Modus.

Bemerkung: Über diese Funktion kann ermittelt werden, unter welchem Host-Modus das DOS-Extender-Programm läuft.

Beispiel:

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <blx286.h>

void main (void)
{
BYTE hostmode;
if(DosGetHostMode(&hostmode) == 0 )
  {
  printf("Der Host-Modus ist ");
  switch(hostmode)
      {
      case BliHostDPMI :
         printf("DPMI");
         break;
      case BliHostVCPI :
         printf("VCPI");
         break;
      case BliHostXMS :
         printf("XMS");
         break;
      default :
         printf("Unbekannt!!");
      }
  }
}
```

DosGetHugeShift()

Funktion: Gibt die Anzahl der Verschiebungen nach links an, die auf 1 angewendet werden soll, um den Inkrementwert des Huge-Selektors zu ermitteln.

Syntax: `USHORT BLXAPI DosGetHugeShift(PUSHORT shiftcount);`

Parameter: shiftcount Zeiger auf Integer-Variable für den Huge-Shift-Wert

Rückgabe: Integerwert mit der Left-Shift-Anzahl.

Bemerkung: Den Huge-Inkrementwert erhält man durch die Anwendung des durch diese Funktion zurückgegebenen Anzahl von Left-Shifts auf 1. Zum Beispiel ergibt eine Left-Shift-Anzahl von 3 ein Huge-Inkrement von 8 (1 << 3 == 8). Dieses Inkrement wird zu dem Basisselektor addiert, der durch DosAllocHuge() zurückgegeben wird. Dadurch erhält man den nächstfolgenden Selektor für einen Huge-Speicherblock.

Beispiel:

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <blx286.h>

void main (void)
{
USHORT shiftcount;
int increment;

/* Zähler für Huge-Shift ermitteln */
if (DosGetHugeShift(&shiftcount) !=0);
  {
  printf("DosGetHugeShift() fehlgeschlagen\n");
  exit(255);
  }
increment = 1 << shiftcount;
printf("Huge-Shift : %04X\n",shiftcount);
printf("Huge-Inkrement : %04X\n",increment);
}
```

DosGetMachineMode()

Funktion: Gibt den aktuellen Rechnermodus an (Real, 286-Protected).

Syntax:

```
USHORT BLXAPI DosGetMachineMode(BYTE __far *mmode);
```

Parameter: Mmode Zeiger auf eine Byte-Variable für den Rechnermodus.

Rückgabe: Anzeigebyte für den Rechnermodus

Bemerkung: DosGetMachineMode() ermittelt den aktuellen Maschinenmodus, entweder Real oder Protected . Da Blinker die Möglichkeit hat, Dualmode-Programme zu erzeugen, die je nach Host-Umgebung im Real Mode oder im Protected Mode laufen, kann DosGetMachineMode() dazu verwendet werden, den augenblicklichen Modus zu bestimmen und entsprechend zu reagieren.

Beispiel:

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <blx286.h>

unsigned char far *VideoBuffer;
```

```
void main (void)
{
BYTE  machinemode;
SEL   selector;

if (DosGetMachineMode(&machinemode) == 0)
  switch (machinemode)
      {
      case MODE_REAL:
         printf ("\nReal Mode - absolute Zeiger ok\n");
         VideoBuffer = MK_FP(0xb800,0);
         break;
      case MODE_PROTECTED:
         printf ("\n286 Protected Mode - viel Speicher\n");
         if (DosMapRealSeg (0xb800, 0x8000L, &selector))
            {
            printf ("\nProblem\n");
            exit (1);
            }
         VideoBuffer = MK_FP(selector,0);
         break;
      default:
         printf ("\nAnderer Modus - interessant\n");
         exit(1);
      }
}
```

Dann:

```
/* Einen '*' in der linken oberen Ecke des Bildschirms anzeigen */
*VideoBuffer[0] = '*';
```

DosGetProtVec()

Funktion: Gibt einen Protected Mode-Zeiger auf einen Protected Mode-Interrupt-Handler zurück

Syntax:
```
USHORT BLXAPI DosGetProtVec( USHORT intno,
                             PINTHAN __far *handler)
```

Parameter:

intno	Interrupt-Nummer
phandler	Zeiger auf einen Funktionszeiger für die zurückgegebene Adresse des Interrupt-Handlers.

Rückgabe: Adresse des Interrupt-Handlers

Bemerkung: DosGetProtVec() ermittelt für den angegebenen Interrupt die Protected Mode Adresse des Protected Mode Interrupt-Handlers.

Beispiel:
```
#include <stdio.h>
#include <blx286.h>

void main (void)
{
INTHAN handler;

if (DosGetProtVec(0x21, &handler) == 0)
  {
  printf("Int 21 Handler f.Protected Mode: %Fp\n",handler);
  }
}
```

Siehe auch: DosGetRealProtVec()
DosGetRealVec()
DosSetProtVec()

DosGetRealProtVec()

Funktion: Gibt sowohl Real Mode als auch Protected Mode Zeiger auf Real Mode und Protected Mode Interrupt-Handler zurück

Syntax:
```
USHORT BLXAPI DosGetRealProtVec( USHORT intno,
                                 PINTHAN __far *prevprot,
                                 REALPTR __far *prevreal);
```

Parameter:

intno	Interrupt-Nummer
prevprot	Zeiger auf einen Funktionszeiger für die zurückgegebene Protected Mode Adresse des Protected Mode Interrupt-Handlers.
prevreal	Zeiger auf einen Funktionszeiger für die zurückgegebene Real Mode Adresse des Real Mode Interrupt-Handlers.

Rückgabe: Adressen der Real Mode und Protected Mode Interrupt-Handler

Bemerkung: DosGetRealProtVec() ermittelt für den angegebenen Interrupt die Real Mode Adresse des Real Mode Interrupt-Handlers und die Protected Mode Adresse des Protected Mode Interrupt-Handlers

Beispiel:

```
#include <stdio.h>
#include <blx286.h>

void main (void)
{
INTHAN prothandler;
REALPTR realhandler;

if (DosGetRealProtVec(0x21, &prothandler, &realhandler) == 0)
  {
  printf("Int 21 Handler für Protected Mode: %Fp\n",prothandler);
  printf("Int 21 Handler für Real Mode: %Fp\n",realhandler);
  }
}
```

Siehe auch: DosGetProtVec()
DosGetRealVec()
DosSetProtVec()
DosSetRealVec()

DosGetRealVec()

Funktion: Gibt einen Real Mode-Zeiger auf einen Real Mode-Interrupt-Handler zurück

Syntax:
```
USHORT BLXAPI DosGetRealVec( USHORT intno,
                             PINTHAN __far *handler)
```

Parameter:

intno	Interrupt-Nummer
handler	Zeiger auf einen Funktionszeiger für die zurückgegebene Real Mode-Adresse des Interrupt-Handlers.

Rückgabe: Adresse des Interrupt-Handlers

Bemerkung: DosGetRealVec() ermittelt für den angegebenen Interrupt die Real Mode Adresse des Real Mode-Interrupt-Handlers.

Beispiel:
```
#include <stdio.h>
#include <blx286.h>

void main (void)
{
REALPTR handler;

if (DosGetRealVec(0x21, &handler) == 0)
  {
  printf("Int 21 Handler für Real mode: %Fp\n",handler);
  }
}
```

Siehe auch: DosGetProtVec()
DosGetRealProtVec()
DosSetRealVec()

DosGetSegBase()

Funktion: Gibt die Basisadresse eines Segmentselektors zurück.

Syntax: `USHORT BLXAPI DosGetSegBase(SEL selector, PULONG baseaddr);`

Parameter:

selector	Vorzeichenloser Wert, der eine gültige Selektornummer darstellt.
baseaddr	Zeiger auf ein Long-Integer Variable für die Basisadresse des Selektors.

Bemerkung: Diese Funktion liefert die lineare Basisadresse des Selektorparameters.

Beispiel:

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <blx286.h>

#define VID_BUF 0xB800
#define VID_SIZE 0x8000l

void main (void)
{
SEL   video_buffer;
ULONG    screen_address;

/* hole Selektor, der dem Bildschirmpuffer entspricht */
if (DosMapRealSeg(VID_BUF,VID_SIZE,&video_buffer) != 0)
  {
  printf("kann Bildschirmsegment nicht ausgeben\n");
  exit(255);
  }
printf("Selektor für Videopuffer %04X\n",video_buffer);
if (DosGetSegBase(video_buffer, &screen_address) == 0)
  {
  printf("Bildschirm-Basisadresse: %08X\n",screen_address);
  }
}
```

DosGetSegDesc()

Funktion: Gibt die Deskriptorstruktur eines Segmentselektors zurück.

Syntax:
```
USHORT BLXAPI DosGetSegDesc( SEL selector,
                             PDESC segdesc);
```

Parameter:

selector	gültige Selektornummer, deren Descriptor angefordert wird.
segdesc	Zeiger auf eine DESC-Struktur (siehe unten), die über DosGetSegDesc() gefüllt werden soll.

Rückgabe: Vereinfachter Segmentdeskriptor mit der Basisadresse, der Länge und den Attributen des Segments, so wie sie in der Headerdatei blx286.h definiert sind.

Bemerkung: Über diese Funktion können alle Informationen zu einem bestimmten Selektor ermittelt werden.

Beispiel:
```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <blx286.h>

void main (void)
{
DESC     segdesc;
if (DosGetSegDesc (SELECTOROF(main),&segdesc) == 0)
  {
  printf ("Basisadresse des aktuellen Code-Segments ist %08Xh\n",
      segdesc.segbase);
  printf ("Länge des aktuellen Code-Segments ist%08Xh\n",
      segdesc.seglen);
  printf ("Das aktuelle Code-Segment hat die Attribute %04Xh\n",
      segdesc.segattrib);
  }
}
```

DosGetSegLimit()

Funktion: Gibt die Größenbegrenzung eines Segmentselektors an.

Syntax:
```
USHORT BLXAPI DosGetSegLimit( SEL selector,
                              PULONG seglimit);
```

Parameter:

selector	gültige Selektornummer
seglimit	Zeiger auf eine Long-Integer Variable für die Segmentbegrenzung des Selektors.

Rückgabe: Die Segmentbegrenzung des angegebenen Selektors.

Bemerkung: Die Segmentbegrenzung stellt den höchsten Offset innerhalb eines Segments dar, der adressiert werden kann.

Beispiel:
```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <blx286.h>

#define VID_BUF 0xB800
#define VID_SIZE 0x8000L

void main (void)
{
SEL      video_buffer;
ULONG       seglimit;

/* Hole Selektor, der dem Bildschirmpuffer entspricht */
if (DosMapRealSeg(VID_BUF,VID_SIZE,&video_buffer) != 0)
  {
  printf("Kann Bildschirmsegment nicht ausgeben\n");
  exit(255);
  }
printf("Selektor für Video-Puffer %04X\n",video_buffer);
if (DosGetSegLimit(video_buffer, &seglimit) == 0)
  {
  /* Rückgabewert müßte in diesem Beispiel 0x7fff sein */
  printf("Bildschirmsegmentgrenze: %08X\n",seglimit);
  }
}
```

DosIsBlinkX()

Funktion: Ermittelt, ob das aktuelle Programm unter dem DOS-Extender von Blinker läuft.

Syntax:
```
USHORT BLXAPI DosIsBlinkX(void);
```

Parameter: keine

Rückgabe: Ein logischer Wert

Bemerkung: Ein logischer Wert (TRUE oder FALSE), der anzeigt, ob das Programm unter dem Blinker-DOS-Extender ausgeführt wird.

Beispiel:
```
#include <stdio.h>
#include <blx286.h>

void main (void)
{
if (DosIsBlinkX())
  {
  printf("Läuft unter dem Blinker-Extender\n");
  }
}
```

DosLockSeg()

Funktion: Sperrt ein Segment im Speicher und verhindert, daß es von der virtuellen Speicherverwaltung des Blinker DOS Extenders ausgelagert wird.

Syntax:
```
USHORT BLXAPI DosLockSeg( SEL selector );
```

Parameter: selector zu sperrender Selektor.

Rückgabe: keine

Bemerkung: DosLockSeg() sperrt den Speicher, der zum übergebenen Selektor gehört, gegen eine Auslagerung auf eine Platte. Diese Funktion sollte dazu verwendet werden, um z.B. Hardware Interrupthandler im Speicher zu halten, da es zum Zeitpunkt des Interruptereignisses nicht möglich sein könnte, DOS aufzurufen und das Segment zurückzuladen. Durch das Sperren eines Selektor wird das zugehörige Segment in den Speicher geladen..

DosLockSeg() sollte auch benutzt werden, um einen Selektor im Speicher zu sperren, bevor er DosCreateDSAlias() oder DosCreateCSAlias() einen Alias erhält. Segmente, die durch diese Funktionen einen Alias erhalten, müssen gesperrt sein, um zu verhindern, daß sie ausgelagert und später wieder an einer anderen linearen Basisadresse zurückgeladen werden. In diesem Falle würde der Alias-Selektor nicht mehr auf den gleichen Speicherbereich zeigen wie der ursprüngliche Alias.

Beispiel:
```
#include <stdio.h>
#include <blx286.h>
```

```
void comminstall(void);
void commhandler(void);
void communinstall(void);

void main(void)
{
if (DosLockSeg(SELECTOROF(commhandler)) == 0)
  {
  printf("Serial interrupt handler locked in memory\n");
  comminstall();   /* Hook serial comms interrupt */
  ...
  communinstall();    /* Unhook comms interrupt */
  if (DosUnlockSeg(SELECTOROF(commhandler)) == 0)
     printf("Comms handler may now be swapped to disk\n");
  }
}
```

Siehe auch: DosUnlockSeg()

DosMapRealSeg()

Funktion: Konvertiert den Adress-Wert eines Real Mode-Paragraphen in einen Protected Mode-Selektor.

Syntax:
```
USHORT BLXAPI DosMapRealSeg(  USHORT paragraph,
                              ULONG size,
                              PSEL selector);
```

Parameter:

paragraph	Eine Real Mode Segmentadresse, z.B. aus der Adresse des Farbbildschirms B800:0000 wird 0xB800.
size	Größe des Speicherbereichs, kann auch größer als 64 KB sein.
selector	Zeiger auf eine Integervariable für den zurückzugebenden Selektor.

Rückgabe: Ein Selektor mit der gleichen Basisadresse wie das Real Mode Segment.

Bemerkung: Mit der Funktion DosMapRealSeg() können Real Mode Segmente auf Protected Mode Selektoren abgebildet werden, um Real Mode Adressen vom Protected Mode aus zugänglich zu machen (z.B. der Real Mode Videopuffer im Segment 0xB800).

Wenn Sie den BIOS-Datenbereich im unteren Speicherbereich im Real Mode Segment 0x0040 vom Protected Mode aus nutzen müssen, verwenden Sie besser die Funktion DosGetBIOSSeg(), um einen Selektor zu erhalten.

DosMapRealSeg() kann einen Speicher NICHT belegen, dazu müssen Sie DosAllocRealSeg() verwenden.

Beispiel:
```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <blx286.h>
```

```
#define VID_BUF 0xB800
#define VID_SIZE 0x8000L

void main (void)
{
SEL video_buffer;

/* Hole Selektor, der dem Bildschirmpuffer entspricht */
if (DosMapRealSeg(VID_BUF,VID_SIZE,&video_buffer) != 0)
  {
  printf("Kann Bildschirmsegment nicht mappen\n");
  exit(255);
  }
printf("Selektor für Video-Puffer %04X\n",video_buffer);
}
```

DosMemAvail()

Funktion: Gibt die Gesamtmenge des verfügbaren Erweiterungsspeichers an.

Syntax: `USHORT BLXAPI DosMemAvail(PULONG bytesavail);`

Parameter: bytesavail — Zeiger auf eine Long-Integer Variable für die zurückgegebene Anzahl der verfügbaren Bytes, im Erweiterungsspeicher.

Rückgabe: Die Gesamtmenge des zur Verfügung stehenden Erweiterungsspeichers in Bytes.

Bemerkung: DosMemAvail() gibt die Größe des verfügbaren erweiterten (extended) Speichers an, der mit DosAllocSeg() oder DosAllocHuge() belegt werden kann.

Beachten Sie, daß ein durch DosFreeSeg() freigegebener Speicher nur an den DOS Extender zurückgegeben wird und nicht an DOS, bevor das Programm beendet wird. Daher kann DosMemAvail() die Zunahme an Speicher nicht erkennen, obwohl DosAllocSeg() den Speicher wieder neu belegen kann.

Beispiel:

```
#include <stdio.h>
#include <blx286.h>

void main (void)
{
ULONG     ext_mem_avail;
if (DosMemAvail(&ext_mem_avail) != 0)
  {
  printf("Freier Extended-Speicher nicht ermittelbar.\n");
  exit(255);
  }
printf("Freier Extended-Speicher ist %08X bytes\n",ext_mem_avail);
}
```

Siehe auch: DosAllocHuge()
DosAllocSeg()
DosRealAvail()

DosProtToReal()

Funktion: Umwandlung eines Protected Mode Pointers in einen entsprechenden Real Mode Pointer.

Syntax: `REALPTR BLXAPI DosProtToReal(void __far * pmptr);`

Parameter: Pmptr Protected Mode Pointer.

Rückgabe: Ein Real Mode Pointer oder NULL, falls die Basisadresse des Protected Mode Pointers oberhalb von 1 MB liegt.

Bemerkung: DosProtToReal() gibt einen Real Mode Pointer zurück, der einem Protected Mode Pointer entspricht.

Beispiel:

```
#include <stdio.h>
#include <blx286.h>

void main(void)
{
void *fp;
REALPTR rmptr;

fp = DosRealToProt ((REALPTR) MK_FP (0x40,0));
if (fp == NULL)
  printf ("Pointer zum BIOS Datenbereich konnte nicht abgebildet werden");
else
  {
  printf ("PM Pointer zum BIOS Datenbereich : %p\n\n",fp);
  if ((rmptr = DosProtToReal(fp)) == NULL)
     printf ("Pointer war im Extended Speicherbereich");
  else
     printf ("RM Pointer zum BIOS Datenbereich : %p\n\n", rmptr);
  }
}
```

Siehe auch: DosRealToProt()

DosRealAvail()

Funktion: Gibt die Gesamtmenge des verfügbaren Real Mode-Speichers zurück.

Syntax:
```
USHORT BLXAPI DosRealAvail(PULONG bytesavail);
```

Parameter: bytesavail Zeiger auf eine Long-Integer Variable, die die Anzahl der Bytes des verfügbaren Real Mode Speichers aufnimmt.

Rückgabe: Die Gesamtmenge an verfügbarem Real Mode-Speicher in Bytes.

Bemerkung: Über DosRealAvail() kann die Speichermenge (in Bytes) festgestellt werden, die zur Belegung durch die Funktion DosAllocRealSeg() bereitsteht.

Beachten Sie, daß ein durch DosFreeRealSeg() freigegebener Speicher nur an den DOS Extender zurückgegeben wird und nicht an DOS, bevor das Programm beendet wird. Daher kann DosRealAvail() die Zunahme an Speicher nicht erkennen, obwohl DosAllocRealSeg() den Speicher wieder neu belegen kann.

Beispiel:
```
#include <stdio.h>
#include <blx286.h>

void main (void)
{
  ULONG vmfree, xmfree, rmfree;
  if (DosVMAvail(&vmfree) == 0)
     printf("%08X Bytes virtueller Speicher sind frei\n",
            vmfree);
  if (DosMemAvail(&xmfree) == 0)
     printf("%08X Bytes Erweiterungsspeicher sind frei\n",
            xmfree);
  if (DosRealAvail(&rmfree) == 0)
     printf("%08X Bytes konventioneller Speicher frei\n",
            rmfree);
}
```

Siehe auch: DosMemAvail()
DosAllocRealSeg()

DosRealFarCall()

Funktion: Startet einen Far-Aufruf an eine Real Mode-Adresse

Syntax:
```
USHORT _far _cdecl DosRealFarCall( REALPTR realfn,
                                   PREGS regs,
                                   REALPTR reserved,
                                   USHORT wordcount,
                                   ... )
```

Parameter:

realfn	Real Mode-Adresse für Aufruf
regs	Zeiger auf die Registerstruktur REGS16, die die Registerwerte für die Übergabe und Rückgabe enthält
reserved	Reserviert: Muß Null sein
wordcount	Anzahl der Word Argumente, die für die Übergabe an realfn auf den Stack gelegt werden. Sie kann auch Null sein für keine Argumente, und ein negativer Wert erzeugt eine Übergabe nach der Pascal-Aufrufkonvention.
...	Argumente zu realfn, wenn wordcount <> 0

Rückgabe: Registerwerte in der REGS16 Struktur

Bemerkung: DosRealFarCall() ermöglicht, daß der Real Mode-Code über einen Far-Aufruf vom Protected Mode aus aufgerufen werden kann. Der Real Mode-Code muß über eine Far-Return Anweisung zurückkehren. Argumente können über den Stack oder in Registern über die Struktur REGS16 übergeben werden. Rückgabewerte werden über die Struktur REGS16 geliefert.

Beispiel:
```
#include <stdio.h>
#include <blx286.h>

void main (void)
{
REGS16      regs;
BYTE     far *xmsdriver;

printf("Programm versucht, die Versionsnummer des\n");
printf("XMS-Treibers über DosRealIntr() und\n");
printf("DosRealFarCall() auszugeben.\n\n");
/* wichtig: Struktur auf Null setzen*/
memset(&regs, 0, sizeof(REGS16));
regs.ax = 0x4300;
if (DosRealIntr(0x2f, &regs, 0L, 0) == 0)
/* xms installation check */
  if ((regs.ax & 0xff)==0x80)
      {
      printf("XMS-Trteiber ist installiert \n");
      /* Treibereintrittspunkt */
      regs.ax = 0x4310;
      if (DosRealIntr(0x2f, &regs, 0L, 0) == 0)
          {
```

```
            xmsdriver = MK_FP(regs.es, regs.bx);
            /* Real Mode-Eintrittspunkt /
            printf("XMS-Treiber Real Mode-Eintrittspunkt : %Fp\n",
               xmsdriver);
            /* Register auf 0 setzen */
            memset(&regs, 0, sizeof(REGS16));
            /* Hole Versionsfunktion */
            regs.ax = 0;
            if (DosRealFarCall((REALPTR)xmsdriver,
               &regs, 0L, 0) == 0)
               if(regs.ax != 0)
                  printf("XMS-Treiber-Version: %04X\n", regs.ax);
               else
                  printf("Abfrage der XMS-Version fehlgeschlagen\n");
            else
               printf("DosRealFarCall() fehlgeschlagen.\n");
            }
         else
            printf("DosRealIntr() fehlgeschlagen.\n");
         }
      else
         printf("XMS-Treiber nicht installiert.\n");
   else
      printf("DosRealIntr()-Aufruf fehlgeschlagen.\n");
   }
```

Siehe auch: DosRealIntr()

DosRealIntr()

Funktion: Erzeugt einen Real Mode-Interrupt vom Protected Mode aus.

Syntax:
```
USHORT BLXAPI DosRealIntr( USHORT intno,
                           PREGS regs,
                           ULONG reserved
                           USHORT wordcount,
                           ...);
```

Parameter:

intno	Real Mode Interrupt-Nummer.
inreg	Zeiger auf die Struktur REGS16, die Registerwerte für die Übergabe und Rückgabe enthält.
reserved	Reserviert: muß Null sein.
wordcount	Anzahl der Word Argumente, die für die Übergabe an den Interrupthandler auf den Stack gelegt werden. Sie kann auch Null sein für keine Argumente, und ein negativer Wert erzeugt eine Übergabe nach der Pascal-Aufrufkonvention.
...	Argumente für den Interrupthandler, wenn wordcount <> 0

Rückgabe: Registerwerte in der REGS16 Struktur

Bemerkung: Über DosRealIntr() wird ein Real Mode-Interrupt vom Protected Mode aus erzeugt. Normalerweise ist es nicht notwendig, diese Funktion aufzurufen, weil Blinker die INT 21h- und BIOS-Aufrufe im Protected Mode zur Verfügung stellt. Aufrufe jedoch, die durch DOS-Erweiterungen wie Netzwerktreiber oder speicherresidente Programme geliefert werden, oder auch undokumentierte DOS-Aufrufe müssen über DosRealIntr() aufgerufen werden.

Beispiel:

```
#include <stdio.h>
#include <blx286.h>

void main (void) /* Hinweis: Dieses Beispiel könnte auch */
                 /* direkt über INT 10h ausgeführt werden. */
{
REGS16   regs;

regs.ax = 0x0f00;
if (DosRealIntr (0x10, &regs, 0L, 0) == 0)
  {
  printf ("Aktueller Videomode: %02X\n",
            regs.ax && 0xFF);
  printf ("Aktuelle Videospalten: %02X\n",
            regs.ax >> 8);
  }
else
  printf ("Aufruf des Videomode fehlgeschlagen\n");
}
```

Siehe auch: DosRealFarCall()

DosReallocHuge()

Funktion: Verändert die Größe eines Erweiterungsspeicherblockes auf bis zu 16 MB.

Syntax:

```
USHORT BLXAPI DosReallocHuge(USHORT segcount,
                             USHORT lastsize,
                             SEL selector);
```

Parameter:

segcount	NeueAnzahl der zu belegenden vollen 64-KB-Segmente.
lastsize	Anzahl der Bytes im letzten Segment.
selector	von DosAllocHuge() zurückgegebener Selektor, als der Speicherblock zu ersten mal belegt wurde.

Rückgabe: Keine

Bemerkung: DosReallocHuge() verändert die Größe eines Speicherblocks, der ursprünglich mit DosAllocHuge angelegt wurde.

Es ist nicht möglich, die Größe von Huge-Blöcken so zu verändern, daß diese größer als ‘segmax+1’ 64-KB-Blöcke sind (wie im ursprünglichen Aufruf von DosAllocHuge() angegeben).

Beispiel:

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <blx286.h>

void main (void)
{
SEL selector;

/* Belegt einen 1-MB-Block (16*64KB) */
/* mit Erweiterungsmöglichkeit auf 2 MB (32*64KB) */
if (DosAllocHuge(16,0,&selector,32, 0) != 0)
  {
  printf("DosAllocHuge fehlgeschlagen");
  exit(255);
  }
/* Versuch, Block auf 1,5 MB (24*64KB) zu erweitern */
if (DosReallocHuge(24,0,selector) != 0)
  {
  printf("DosReallocHuge fehlgeschlagen\n");
  exit(254);
  }
printf("Block erfolgreich auf 1.5 MB erweitert\n");
DosFreeSeg(selector);
}
```

DosReallocSeg()

Funktion: Verändert die Größe eines Erweiterungsspeicherblockes auf bis zu 64 KB.

Syntax:

```
USHORT BLXAPI DosReallocSeg( USHORT size,
                             SEL selector);
```

Parameter:

size	Neue Segmentgröße in Bytes (0 = 65.536).
selector	Selektor vom vorhergehenden Aufruf von DosAllocSeg(), mit dem der Block ursprünglich angelegt wurde.

Rückgabe: Keine

Bemerkung: DosReallocSeg() verändert die Größe eines zuvor über DosAllocSeg() belegten Speicherblockes. Die neue Größe eines Blocks darf 64KB nicht überschreiten (verwenden Sie DosAllocHuge() / DosReallocHuge() für größere Speicherblöcke).

Beispiel:

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <blx286.h>
```

```
void main (void)
{
SEL selector;

/* Belegt einen 4-KB-Block */
if (DosAllocSeg(4096, &selector, 0) != 0)
  {
  printf("DosAllocSeg fehlgeschlagen \n");
  exit(255);
  }
printf("4-KB-Block belegt bei: %04X\n",selector);
/* Erweitert den Block auf 8 KB */
if (DosReallocSeg(8192, &selector) != 0)
  {
  printf("DosReallocSeg fehlgeschlagen\n");
  exit(254);
  }
printf("Block erfolgreich auf 8 KB erweitert.\n");
DosFreeSeg(sel);
}
```

DosRealToProt()

Funktion: Umwandlung eines Real Mode Pointers in einen entsprechenden Protected Mode Pointer.

Syntax: `Void __far * BLXAPI DosRealToProt(REALPTR rmptr);`

Parameter: Rmptr Real Mode Pointer.

Rückgabe: Ein Protected Mode Pointer.

Bemerkung: DosRealToProt() gibt einenProtected Mode Pointer zurück, der einem Real Mode Pointer entspricht.

Beispiel:

```
#include <stdio.h>
#include <blx286.h>

void main(void)
{
void *fp;
REALPTR rmptr;

fp = DosRealToProt ((REALPTR) MK_FP (0x40,0));
if (fp == NULL)
  printf ("Kann Pointer auf BIOS Datenbereich nicht umwandeln");
else
  {
  printf ("PM Pointer auf BIOS Datenbereich : %p\n\n",fp);
  if ((rmptr = DosProtToReal(fp)) == NULL)
     printf ("Pointer war im Extended Speicher");
```

```
  else
     printf ("RM pointer auf BIOS Datenbereich : %p\n\n", rmptr);
  }
}
```

Siehe auch: __DosProtToReal()

DosSetExceptionHandler()

Funktion: Installiert einen Protected Mode-Ausnahme-Handler

Syntax:
```
USHORT BLXAPI DosSetExceptionHandler( USHORT excno,
                                      PEXCHAN newhandler,
                                      PEXCHAN __far *prevhandler);
```

Parameter:

excno	Nummer der Ausnahme (0-13)
newhandler	Zeiger auf den neuen Handler
prevhandler	Zeiger auf Funktionszeiger, in dem die zurückzugebende Adresse des vorherigen Handlers abgelegt werden soll.

Bemerkung: DosSetExceptionHandler() installiert einen Protected Mode-Handler für Prozessorausnahmen.

Folgende Ausnahmen sind möglich:

```
0x00  Division durch Null
0x01  Debug
0x03  Breakpoint
0x04  Überlauf
0x05  Grenzprüfung
0x06  Ungültiger Opcode
0x07  287/387-Coprozessor nicht verfügbar
0x08  Doppelter Fehler
0x0A  Ungültiges Task-Schaltersegment
0x0B  Segment nicht vorhanden
0x0C  Stackfehler
0x0D  Allgemeine Schutzverletzung
```

Beispiel:
```
#include <stdio.h>
#include <stdlib.h>
#include <blx286.h>

PEXCHAN prevhandler;

void printstr (char *s)      /* C Screen I/O Funktionen können */
                           /* hier nicht verwendet werden */
  {
  char c;
  while (c = *(s++))         /* bis Ende des Strings */
     {
```

```
      _asm mov ah,0eh
      _asm mov al,c
      _asm int 10h        /* Display teletype */
      }
  }
void _interrupt far gphandler(EXCEP_FRAME ef)
  {
  char msg [80];
  printstr ("Ersatz GPF Handler\n\r");
  printstr ("----------------------\n\r");
  sprintf (msg,"Errorcode = %04x\n\r",ef.error_code);
  printstr (msg);
  sprintf (msg," CS:IP = %04x:%04x\n\r", ef.ret_cs, ef.ret_ip);
  printstr (msg);
  DosSetExceptionHandler (0x0d,prevhandler,NULL);
  exit (255);
}
void main (void)
{
PEXCHAN newhandler;

newhandler = gphandler;
if (DosSetExceptionHandler (0x0d, newhandler,&prevhandler) == 0)
  {
  *((USHORT far *) 0x22222222L) = 1;   /* Eine Exception erzeugen*/
  printf("Irgendwie ist keine Exception aufgetreten\n");
  DosSetExceptionHandler (0x0d,prevhandler,NULL);
  }
}
```

Siehe auch: DosGetExceptionHandler()

DosSetProtVec()

Funktion: Installiert einen Protected Mode-Interrupt-Handler

Syntax:
```
USHORT BLXAPI DosSetProtVec( USHORT intno,
                             PINTHAN newhandler,
                             PINTHAN __far * prevhandler);
```

Parameter:

intno	Interrupt-Nummer
newhandler	Zeiger auf den neuen Handler
prevhandler	Zeiger auf Funktionszeiger, in dem die zurückzugebende Adresse des vorherigen Handlers abgelegt werden soll.

Rückgabe: Adresse des vorherigen Interrupt-Handlers

Bemerkung: DosSetProtVec() installiert einen Protected Mode-Interrupt-Handler für den angegebenen Protected Mode-Interrupt. DosSetProtVec() liefert ebenfalls einen Zeiger auf den vorhergehenden Interrupt-Handler.

Beispiel:
```
#include <stdio.h>
#include <dos.h>
#include <blx286.h>

USHORT ticks;
PINTHAN prevhandler;

void interrupt far timer1c(REGS16 r);
{
  ticks++;
  _chain_intr(&prevhandler);
}

void main (void)
{
PINTHAN  newhandler;
int         cnt0,cnt1;

ticks = 0;
newhandler = timer1c;
if (DosSetProtVec(0x1c, newhandler, &prevhandler) == 0)
  {
  printf("Benutzerdefinierter Timer erfolgreich eingebunden.\n");
  printf("einige Sekunden warten... \n");
  for (cnt0 = 0; cnt0 < 500; cnt0++)
     for (cnt1 = 0; cnt1 < 32000; cnt1++);
  if (DosSetProtVec(0x1c, prevhandler, NULL) == 0)
     printf("Timer-Handler deinstalliert\n");
  else
     printf("Timer-Handler nicht deinstallierbar!\n");
  printf("Anzahl der Takte im Protected Mode : %04X\n",ticks);
  }
```

```
else
  {
  printf("Timer-Interrupthandler konnte nicht verändert werden.\n");
  }
}
```

Siehe auch: DosGetProtVec()
DosGetRealProtVec()
DosSetRealVec()

DosSetRealVec()

Funktion: Installiert einen Real Mode-Interrupt-Handler

Syntax:
```
USHORT BLXAPI DosSetRealVec( USHORT intno,
                             REALPTR newhandler,
                             REALPTR __far *prevhandler);
```

Parameter:

intno	Interrupt-Nummer
newhandler	Zeiger auf den neuen Handler
prevhandler	Zeiger auf Funktionszeiger, in dem die zurückzugebende Adresse des vorherigen Handlers abgelegt werden soll.

Rückgabe: Adresse des vorherigen Interrupt-Handlers

Bemerkung: DosSetRealVec() installiert einen Real Mode-Interrupt-Handler für den angegebenen Real Mode-Interrupt. DosSetRealVec() liefert auch einen Zeiger auf den vorhergehenden Interrupt-Handler.

Beispiele für den Einsatz dieser Funktion finden Sie in den Dateien INTMAIN.C, INTHAND.ASM und INTMAIN.LNK im C Unterverzeichnis von Blinker.

Beispiel: `Siehe INTMAIN.C`

Siehe auch: DosGetRealVec()
DosGetRealProtVec()
DosSetProtVec()

DosSetSegAttrib()

Funktion: Setzt die Segmentattribute für einen Selektor

Syntax: `USHORT BLXAPI DosSetSegAttrib(SEL selector, USHORT attr);`

Parameter:

selector	Selektor des Segments
attr	neue Attribute

Rückgabe: keine

Bemerkung: Über DosSetSegAttrib() kann der Typ eines Segments von Code auf Daten und umgekehrt gewechselt werden. Außerdem können die Segmentattribute Lesen/Schreiben/Ausführen gesetzt werden. Die möglichen Attribute sind in BLX286.H deklariert.

Anders als DosCreateCSAlias() und DosCreateDSAlias(), erzeugt DosSetSegAttrib() KEINEN neuen Deskriptor. Stattdessen wird der vorhandene Deskriptor direkt modifiziert.

Beispiel:

```
#include <stdio.h>
#include <blx286.h>

unsigned char far array[] = { 1, 2, 3, 4};

void main (void)
{
/* Segment des Arrays mit Attribut NUR LESEN versehen */
if (DosSetSegAttr(SELECTOROF(array[0]), DATA16_NOWRITE) == 0)
  {
  printf("Segment mit Array ist nun nur lesbar (read only).\n");
  }
/* Zurücknahme des Attributs */
if (DosSetSegAttr(SELECTOROF(array[0]), DATA16) == 0)
  {
  printf("Segment kann jetzt beschrieben und gelesen
         werden.\n");
  }
}
```

Siehe auch: DosVerifyAccess()

DosUnlockSeg()

Funktion: Entsperrt ein Speichersegment, das zuvor über DosLockSeg() gesperrt worden ist.

Syntax:
```
USHORT BLXAPI DosUnlockSeg( SEL selector );
```

Parameter: selector Selektor, der zuvor über DosLockSeg() gesperrt worden ist.

Rückgabe: keine

Bemerkung: DosUnLockSeg() entsperrt den Speicher, der zum übergebenen Selektor gehört. Dadurch kann der Speicher wieder auf eine Festplatte ausgelagert werden. Mit dieser Funktion kann man Hardware Interrupt-Handler freigeben, wenn sie nicht mehr aktiv sind, und auch jeden Selektor entsperren, dessen Alias freigegeben worden ist.

Beispiel:
```
#include <stdio.h>
#include <blx286.h>

void comminstall(void);
void commhandler(void);
void communinstall(void);

void main(void)
{
if (DosLockSeg(SELECTOROF(commhandler)) == 0)
  {
  printf("Interrupt Handler für serielle Schnittstelle im Speicher
gesperrt\n");
  comminstall();   /* Interrupt-Handler für ser. Komm. installieren */
  ...
  communinstall();    /* Interrupt-Handler deinstallieren */
  if (DosUnlockSeg(SELECTOROF(commhandler)) == 0)
     printf("Interrupt-Handler kann nun auf Platte ausgelagert werden\n");
  }
}
```

Siehe auch: DosLockSeg()

DosVerifyAccess()

Funktion: Gibt die für einen Selektor gesetzten Flags Lesen/Schreiben/Code/Daten zurück

Syntax: `USHORT BLXAPI DosVerifyAccess(SEL selector, PUSHORT flags);`

Parameter:

selector	Selektor oder irgendein vorzeichenloser Wert
flags	Zeiger auf eine Integer-Variable für die zurückzugebenden Flags

Rückgabe: Die Flags des Segments

Bemerkung: Über DosVerifyAccess() kann ermittelt werden, ob das Selektorargument einen gültigen Selektor enthält, und wenn ja, von welchem Typ er ist (Code oder Daten) und welche Zugriffsrechte er hat (nur lesbar oder beschreibbar).

Beispiel:

```
#include <stdio.h>
#include <blx286.h>

void DumpFlags(SEL sel)
{
  USHORT flags;
  if (DosVerifyAccess(sel, &flags) == 0)
     if (flags & SEL_VALID)     /* falls gültiger Selektor */
        {
        if (flags & SEL_WRITABLE) printf("WRITABLE ");
        if (flags & SEL_READABLE) printf("READABLE ");
        printf("%s\n", (flags & SEL_CODE) ? "CODE" :"DATA");
        }
     else
        {
        printf("KEIN SELEKTOR\n");
        }
}

void main (void)
{
SEL   selector;

if (DosLockSeg(SELECTOROF(main)) == 0)
  if (DosCreateDSAlias (SELECTOROF (main), &selector) == 0)
     {
     printf ("Code-Selektor für main() ist %04X\n",
               SELECTOROF(main));
     printf ("Datenselektor ist %04X\n", selector);
     printf ("Codeselektor-Flags : ");
     DumpFlags(SELECTOROF(main));
     printf ("Datenselektor-Flags : ");
     DumpFlags(selector);
     }
}
```

Siehe auch:

DosSetSegAttrib()

DosVMAvail()

Funktion: Gibt die Größe des verfügbaren virtuellen Speichers an

Syntax: `USHORT DosVMAvail(PULONG bytes);`

Parameter:

bytes	Zeiger auf eine Long-Integer Variable für die verfügbaren virtuellen Speichers	Anzahl der Bytes des

Rückgabe: Gesamter verfügbarer virtueller Speicher in Bytes

Bemerkung: DosVMAvail() liefert die ungefähre Anzahl der Bytes des virtuellen Speichers zurück, die dem Protected Mode-Programm zur Verfügung stehen.

Beispiel:

```
#include <stdio.h>
#include <blx286.h>

void main (void)
{
ULONG vmfree, xmfree, rmfree;

if (DosVMAvail(&vmfree) == 0)
  printf("%08X Bytes virtueller Speicher frei\n",vmfree);
if (DosMemAvail(&xmfree) == 0)
  printf("%08X Bytes Erweiterungsspeicher frei\n",xmfree);
if (DosRealAvail(&rmfree) == 0)
  printf("%08X Bytes konventioneller Speicher frei\n",rmfree);
}
```

Siehe auch: DosMemAvail()
DosRealAvail()

SWPADDENV()

Funktion: Vergrößert den Umgebungsblock des untergeordneten Programms.

Syntax:
```
niWert = SWPADDENV(niIncrease)
```

Parameter: niIncrease Anzahl Bytes, die der Umgebung hinzugefügt werden sollen.

Rückgabe: Die vorhergehende Einstellung

Bemerkung: Wenn die DOS-Funktion EXEC einen Prozeß startet (SWPRUNCMD() verwendet diese DOS-Funktion), dann wird die für das untergeordnete Programm erzeugte Umgebung standardmäßig abgeschnitten, so daß , wenn überhaupt, sehr wenig freier Speicherplatz zur Verfügung steht.

Mit dieser Funktion kann die Umgebung des untergeordneten Programms um die Bytesanzahl vergrößert werden, die durch den Parameter übergeben wurde. Das kann sich als notwendig erweisen, wenn die Umgebungsvariablen des untergeordneten Programms verändert oder welche hinzugefügt werden müssen. Maximal kann ein Umgebungsblock 32 KB groß werden.

Beachten Sie, daß diese Funktion die Umgebung des untergeordneten Prozesses vergrößert, während dieser Prozeß ausgeführt wird. Die Umgebung des übergeordneten Prozesses wird nicht verändert.

Die Umgebung des untergeordneten Programms wird gelöscht, sobald das untergeordnete Programm beendet ist.

Beispiel:
```
// Sicherstellen, daß 128 Bytes in der Umgebung frei sind
OldSize = SWPADDENV(128)
```

Siehe auch: SWPSETENV() / SWPSETENVBAS()

SWPADDSTR() / SWPADDSTRBAS()

Funktion: Rückgabe eines String an das vorhergehende übergeordnete Programm

Syntax: `nivalue = SWPADDSTR(cPID, cString)`

Parameter:

cPID	ID des übergeordneten Programms.
cString	String für Rückgabe an übergeordnetes Programm.

Rückgabe: Ein logischer Wert

Bemerkung: Über SWPADDSTR() kann ein String von bis zu 127 Bytes vom untergeordneten an das übergeordnete Programm übergeben werden. Der Rückgabestring darf keine NULL-Zeichen enthalten, weil das erste NULL-Zeichen als Stringende interpretiert wird.

Der Rückgabestring sollte zunächst mit der Funktion SWPSETSTR() erstellt werden. Mit SWPADDSTR() können dann weitere Zeichen an das Ende des Strings angefügt werden.

Das übergeordnete Programm, das den String erhalten soll, wird über seine Programm-ID identifiziert (wird im übergeordneten Programm über die Funktion SWPSETPID() gesetzt). Das übergeordnete Programm holt sich den String über die Funktion SWPGETSTR().

Beispiel:

```
success := SWPSETSTR("TEST","Zwei")
success := SWPADDSTR("TEST"," Teile")
```

liefert den String "Zwei Teile" an das Programm mit der Programm-ID "TEST".

Hinweise : Der Rückgabewert ist ein logischer Wert, der Erfolg oder Mißerfolg der Operation anzeigt. Hierbei sind drei Fehlerquellen möglich:

- 1 - Der Kern des angegebenen übergeordneten Programms befindet sich zur Zeit nicht im Speicher.
- 2 - Der Rückgabestring ist länger als 127 Bytes.
- 3 - Das übergeordnete Programm wurde mit einer Version von Blinker gelinkt, die älter als 2.10 ist.

Siehe auch:

SWPCURDIR()

Funktion: Speichern des aktuellen Verzeichnisses ein-/ausschalten

Syntax: `lWert = SWPCURDIR(lFlag)`

Parameter: lValue logischer Wert für ein- / ausgeschaltet.

Rückgabe: Die vorherige Einstellung

Standard: Eingeschaltet

Bemerkung: Diese Funktion erlaubt das Ein- und Ausschalten der Wiederherstellung des aktuellen Laufwerks und des aktuellen Pfades nach der Swap-Aktion. Ist diese Funktion aktiv, so wird das Laufwerk und Verzeichnis nach der Swap-Aktion wieder so hergestellt, wie vor dem Eintritt in die Swap-Routine.

Beispiel:
```
SWPCURDIR(.T.)      // Laufwerk und Verzeichnis beim Swap speichern
```

SWPDISMSG()

Funktion: Ein-Ausschalten der Ausgabe einer Nachricht nach dem Swap

Syntax: `lWert = SWPDISMSG(lWert)`

Parameter: lWert logischer Wert für ein- / ausgeschaltet.

Rückgabe: Die vorherige Einstellung

Standard: Ausgeschaltet

Bemerkung: Ein-Ausschalten der Ausgabe des Textes ‘Beliebige Taste, um forzufahren’, bevor zu dem Eltern-Prozeß zurückgekehrt wird. Der Swap-Kernel gibt diese Nachricht dann aus und wartet anschließend auf einen Tastendruck, um den Eltern-Prozeß zu laden, sollte der Schalter auf TRUE gestellt worden sein, andernfalls wird keine Aktion ausgeführt.

Beispiel:
```
* Meldung anzeigen mit anschließendem Warten auf einen
* Tastendruck, nachdem der Befehl DIR ausgeführt wurde
SWPDISMSG(.T.)
SWPRUNCMD("DIR",0,"","")
```

Siehe auch: SWPGETKEY()

SWPEMS320()

Funktion: Swapper auf Verwendung von EMS 3.2 Aufrufen begrenzen

Syntax: `lWert = SWPEMS320(lWert)`

Parameter: lWert logischer Wert für ein- / ausgeschaltet.

Rückgabe: Die vorige Einstellung

Bemerkung: Bei einigen speziellen OEM EMS-Emulatoren sind Probleme mit dem Einsatz von EMS 4.0 Funktionen aufgetreten, die anscheinend in den betreffenden Treibern unsauber implementiert wurden. Mit dieser Funktion wird der Swapper gezwungen, nur EMS 3.2 Funktionen zu verwenden.

Beispiel:
```
SWPEMS320(.T.)   // nur EMS 3.2 Aufrufe verwenden.
```

Hinweis: In den meisten Fällen wird diese Funktion nicht benötigt.

SWPERRLEV()

Funktion: Den Rückgabewert des Kind-Prozesses ermitteln

Syntax: `niWert = SWPERRLEV()`

Parameter: keine

Rückgabe: Rückgabwert des Kind-Prozesses

Bemerkung: Diese Funktion gibt den Rückgabewert des Kind-Prozesses zurück. Der zurückgegebene Wert ist der Exit-Code des letzten Prozesses, welcher vor dem Kommando-Interpreter (command.com) beendet wurde, der zurück zu dem Swap-Kernel führt. Dieser Wert kann durch Ausführen des Programms von einer DOS-Batchdatei aus bestätigt und auf den vom Programm zurückgegebenen ERRORLEVEL geprüft werden. In dem Fall, daß der Kommando-Interpreter der einzige auszuführende Task war, wird der Rückgabewert dieses Kommando-Interpreters zurückgegeben.

Beispiel:
```
* Auswerung des Errorlevel eines Kind-Prozesses
*
SWPRUNCMD("Kompiliere meinprog -m",0,"","")
IF SWPERRLEV() <> 0
  ? "Kompilieren fehlgeschlagen..."
ELSE
  SWPRUNCMD("BLINKER @LINK.LNK",0,"","")
ENDIF
```

SWPERRMAJ()

Funktion: Ermitteln des Haupt-Fehlercodes des Swap-Systems

Syntax: `niWert = SWPERRMAJ()`

Parameter: keine

Rückgabe: Fehler-Code des Swap-Systems

Bemerkung: Tritt ein Fehler während des Swap-Prozesses auf, so liefert die Funktion SWPRUNCMD() FALSE. SWPERRMAJ() liefert den Haupt-Fehlercode, welcher Aussagen liefert bezüglich der Art des aufgetretenen Fehlers.

Die Haupt- und Unter-Fehlercodes und ihre Bedeutungen sind aufgelistet im Abschnitt 'Swap-Laufzeitfehlermeldungen (1220)' im Kapitel 11, Fehlermeldungen.

Beispiel:

```
* Ermittlung des Haupt-Fehlercodes eines Kind-Prozesses
*
success = SWPRUNCMD("Kompiliere meinprog -m",0,"","")
IF .NOT. success
  ? "Swap fehlgeschlagen..."
  ? "Haupt-Fehlernummer ",SWPERRMAJ()
  ? "Unter-Fehlernummer ",SWPERRMIN()
ELSE
  err_lev = SWPERRLEV()
  IF err_lev = 0
     SWPRUNCMD("BLINKER @myscr",0,"","")
  ELSE
     ?"Kompilieren fehlgeschlagen - Fehler: ",err_lev
  ENDIF
ENDIF
```

Siehe auch: SWPERRMIN()
Swap-Laufzeitfehlermeldungen (1220)

SWPERRMIN()

Funktion: Ermitteln des Unter-Fehlercodes des Swap-Systems

Syntax: `niWert = SWPERRMIN()`

Parameter: keine

Rückgabe: Unter-Fehlercode des Swap-Systems

Bemerkung: Tritt ein Fehler während des Swap-Prozesses auf, so liefert die Funktion SWPRUNCMD() FALSE. SWPERRMIN() liefert den Unter-Fehlercode, welcher Aussagen liefert bezüglich der Art des aufgetretenen Fehlers.

Die Haupt- und Unter-Fehlercodes und ihre Bedeutungen sind aufgelistet im Abschnitt 'Swap-Laufzeitfehlermeldungen (1220)' im Kapitel 11, Fehlermeldungen.

Beispiel:

```
success = SWPRUNCMD("Kompiliere Meinprog -m",0,"","")
IF .NOT. success
  ? "Swap fehlgeschlagen..."
  ? "Haupt-Fehlernummer ",SWPERRMAJ()
  ? "Unter-Fehlernummer ",SWPERRMIN()
ELSE
  err_lev = SWPERRLEV()
  IF err_lev = 0
     SWPRUNCMD("BLINKER @LINK.LNK",0,"","")
  ELSE
     ? "Kompilieren fehlgeschlagen - Fehler: ", err_lev
  ENDIF
ENDIF
```

Siehe auch: SWPERRMAJ()
Swap-Laufzeitfehlermeldungen (1220)

SWPFREEMS()

Funktion: Begrenzt die Verwendung des EMS-Speichers für die Auslagerung

Syntax: `niWert = SWPFREEMS(niWert)`

Parameter: niWert Maximum für EMS in KB

Rückgabe: Die vorhergehende Einstellung

Bemerkung: Über diese Funktion kann die Menge des EMS-Speichers begrenzt werden, die vom Swap-System für die Speicherung des Programm-Abbildes verwendet wird, so daß die angegebene Menge für ein untergeordnetes Programm freigelassen wird.

Beachten Sie bitte, daß diese Funktion nur die Menge des EMS-Speichers beschränkt, die von der Funktion SWPRUNCMD() verwendet wird. Sämtlicher anderer durch die Anwendung genutzter EMS-Speicher wird nicht freigegeben. Die als Parameter angegebene Menge freizulassenden EMS-Speichers muß bereits zu dem Zeitpunkt frei sein, wenn SWPRUNCMD() ausgeführt wird.

Beispiel:
```
SWPUSEEMS(.T)    // Ermögliche die Verwendung von EMS,
 SWPFREEMS(512)     // lasse aber mindestens 512 KB frei,
                  // wenn das möglich ist.
```

Siehe auch: SWPFREXMS()

SWPFREXMS()

Funktion: Begrenzt die Verwendung des XMS-Speichers für die Auslagerung

Syntax: `niWert = SWPFREXMS(niWert)`

Parameter: niWert Maximum für XMS in KB

Rückgabe: Die vorhergehende Einstellung

Bemerkung: Über diese Funktion kann die Menge des XMS-Speichers begrenzt werden, die vom Swap-System für die Speicherung des Programm-Abbildes verwendet wird, so daß die angegebene Menge für ein untergeordnetes Programm freigelassen wird

Beachten Sie bitte, daß diese Funktion nur die Menge des XMS-Speichers beschränkt, die von der Funktion SWPRUNCMD() verwendet wird. Sämtlicher anderer durch die Anwendung genutzter XMS-Speicher wird nicht freigegeben. Die als Parameter angegebene Menge freizulassenden XMS-Speichers muß bereits zu dem Zeitpunkt frei sein, wenn SWPRUNCMD() ausgeführt wird.

Beispiel:
```
SWPUSEXMS(TRUE)  // Ermögliche Verwendung von XMS,
 SWPFREXMS(512)     // lasse aber mindestens 512 KB frei,
                  // wenn das möglich ist.
```

Siehe auch: SWPFREEMS()

SWPGETKEY()

Funktion: Warten auf Tastendruck ein-/ausschalten, bevor zum Eltern-Prozeß zurückgekehrt wird

Syntax: `lWert = SWPGETKEY(lWert)`

Parameter: lWert logischer Wert für ein-/ausgeschaltet

Rückgabe: Die vorherige Einstellung

Standard: Ausgeschaltet

Bemerkung: Diese Funktion schaltet das Warten auf einen Tastendruck, bevor zum Eltern-Prozeß zurückgekehrt wird, ein oder aus. Der Video-Modus wird nach dem Tastendruck wieder auf den ursprünglichen Modus gesetzt, sofern des Sichern des Video-Modus eingeschaltet ist. Diese Funktion kann z.B. dazu verwandt werden, auf einen Tastendruck zu warten, während ein grafisches Bild angezeigt wird.

Beispiel:
```
* Einstellen des Wartens auf einen Tastendruck
*
SWPGETKEY(.T.)      && Warte auf Tastendruck
SWPVIDMDE(.T.)      && Videomode sichern
SWPRUNCMD("ZEIGBILD SCHOEN.IMG",0,"","")
```

Siehe auch: SWPDISMSG()

SWPGETPID() / SWPGETPIDBAS()

Funktion: Ermitteln, ob angegebenes Programm das Eltern-Programm zum aktuellen Programm ist

Syntax: `lWert = SWPGETPID(cIdString)`

Parameter: cIdString Programm ID String

Rückgabe: Logischer Wert, der angibt, ob der übergebene String als Programm-ID durch den Eltern-Prozeß gesetzt worden ist.

Bemerkung: Ermittelt, ob cldString ein Eltern-Prozeß (jeder vorherigen Stufe) der aktuell ausgeführten Applikation ist. cldString kann nur 31 Zeichen oder kürzer sein, andernfalls gibt die Funktion FALSE zurück. Der String muß exakt übereinstimmen mit einer ID, die durch SWPSETPID() gesetzt worden ist. Dies kann dann sinnvoll sein, um um verhindern, daß ein und dieselbe Applikation mehrfach hintereinander gestartet werden kann.

Beispiel:
```
* Ermittle, ob die aktuelle Applikation schon
* ausgeführt wird
*
IdStr = "Test Programm"
IF SWPGETPID(IdStr)
  ? IdStr + " läuft bereits!"
  QUIT
ENDIF
SWPSETPID(IdStr)
```

Siehe auch: SWPADDSTR() / SWPADDSTRBAS()
SWPGETSTR() / SWPGETSTRBAS()
SWPSETPID() / SWPSETPIDBAS()
SWPSETSTR() / SWPSETSTRBAS()

SWPGETSTR() / SWPGETSTRBAS()

Funktion: Empfängt den vom untergeordneten Programm über die Funktion SWPSETSTR() gesendeten String

Syntax:
```
cstring = SWPGETSTR()
```

Parameter: keine

Rückgabe: Ein String

Bemerkung: SWPGETSTR() wird im übergeordneten Programm eingesetzt, um den vom untergeordneten Programm gesendeten String zu empfangen.

Beispiel:
```
// Übergeordnetes Programm
SWPSETPID("PARENT")
SWPRUNCMD("child",0,"","")
string = SWPGETSTR()
? "Der Rückgabestring lautet : ",string // 'Hallo'
// eof parent

// Child program
if SWPGETPID("PARENT")
  SWPSETSTR("PARENT","Hallo")
endif
quit
// eof child
```

Siehe auch: SWPADDSTR() / SWPADDSTRBAS()
SWPGETPID() / SWPGETPIDBAS()
SWPSETPID() / SWPSETPIDBAS()
SWPSETSTR() / SWPSETSTRBAS()

SWPKEYBRD() / SWPKEYBRDBAS()

Funktion: Übergibt Tastendrücke an den untergeordneten Prozeß

Syntax: `niWert = SWPKEYBRD(cKeyString)`

Parameter: cKeyString String aus Buchstaben und Tastaturwerten

Rückgabe: 0 Stringanalyse OK

1 Fehler - Anführungszeichen fehlen. Im Eingabe-string gibt es ein öffnendes Anführungszeichen

2 Fehler - '{' fehlt. Es fehlt entweder '{' oder '}' (Begrenzer für nicht-ASCII-Tasten).

nnirgend ein anderer Wert, der anzeigt, daß die Analyse an der angegebenen Position im Eingabe-String fehlgeschlagen ist.

Bemerkung: Die Funktion SWPKEYBRD ermöglicht es, daß das übergeordnete Programm verschiedene Tastendrücke zu deren Ausführung in den Tastaturpuffer des untergeordneten Programms stellen kann. Die Verwendung dieser Funktion ermöglicht Ihnen die vollständige Kontrolle über die Ausführung des untergeordneten Programms.

SWPKEYBRD() funktioniert mit den meisten untergeordneten Programmen, die die BIOS-Tastaturfunktionen 0,1,10H und 11H zum Lesen der Tastatureingabe verwenden. Programme, die auf die Tastaturhardware direkt zugreifen (wie z.B. MS Word) können die übergebenen Tastendrücke nicht lesen und daher nicht über diese Funktion gesteuert werden.

Maximal können 250 Tastendrücke übergeben werden.

SWPKEYBRD-Parameter

SWPKEYBRD übernimmt einen Stringparameter, der aus folgenden Einzelteilen besteht:

- eingebettete Teilstrings;
- b) In Anführungszeichen eingebettete Teilstrings;
- c) Spezielle Tastenkombinationen und Tasten, die nicht in ASCII-Format angegeben werden können, so wie PgUp und ALT-F1

Eingebettete Teilstrings

Eingebettete Teilstrings enthalten normalerweise ASCII-Zeichen, die auf der Tastatur zu finden sind, wie A-Z, a-z, 0-9, !@#$%^&*() u.s.w. Diese Zeichen werden durch SWPKEYBRD() in die Tastaturcodes umgewandelt, die zu den jeweiligen Tasten gehören (entsprechend der Zeichenlegende auf einer US-Tastatur).

Eingebettete Teilstrings können auch Zeichen enthalten, die es auf der US-Tastatur nicht gibt, wie internationale Zeichen, Box- und Linienzeichen u.s.w. Trifft SWPKEYBRD() auf solch ein Zeichen, dann generiert die Funktion einen Tastencode, der dem Tastencode <ALT-nnn> wntspricht, wobei nnn der ASCII-Wert des fraglichen Zeichens ist.

Anführungszeichen

Eingebettete Teilstrings müssen entweder mit einem einzelnen (‘) oder einem doppelten Anführungszeichen (") beginnen und enden. Anfangs- und Endezeichen müssen jedoch

zueinander passen. Das ermöglicht Ihnen, Anführungszeichen auch innerhalb des Textes zu verwenden:

swpkeybrd("{enter}‘Er sagte zweimal "Hallo"’{enter}")

swpkeybrd(‘{enter}"Drücken Sie jetzt bitte auf das ‘X’!"{enter}’)

Beachten Sie, daß die meisten Compiler das erste und das letzte Anführungszeichen innerhalb des String als Begrenzer betrachten. Der an SWPKEYBRD() übergebene String muß innerhalb eines anderen Paares von Anführungszeichen stehen.

Nicht-ASCII-Tastenkombinationen

Zusätzlich zu dem in Anführungszeichen eingeschlossenen Text kann SWPKEYBRD() auch andere Tastendrücke übergeben, die nicht durch ASCII-Zeichen dargestellt werden können. Viele dieser Tastendrücke können mit folgenden drei Tasten kombiniert werden:

```
Shift, Ctrl und Alt
```

Wird eine solche Shift-State-Taste angegeben, dann sollte sie vor der aktuellen Tastendefinition stehen und von dieser durch ein ‘-’-Zeichen getrennt werden. Zum Beispiel:

```
{Shift-F7}{CTRL-a}{ALT-ESC}
```

Sie können nicht mehr als ein Shift-State gleichzeitig verwenden - BIOS unterstützt dies nicht. Wenn das untergeordnete Programm mehrere Shift-States simultan verwendet, wie z.B. <Ctrl>+<Alt>, <Ctrl>+<Shift>, dann können Sie es nicht über SWPKEYBRD() steuern, da es entweder direkt auf die Tastaturhardware zugreift oder Shift-State-Informationen aus dem BIOS-Datenbereich verwendet, die SWPKEYBRD() nicht zu emulieren versucht.

Folgende Tasten stehen zur Verfügung:

{f1} - {f12}
{shift-f1} - {shift-f12}
{ctrl-f1} - {ctrl-f12}
{alt-f1} - {alt-f12}
{ctrl-a} - {ctrl-z}
{alt-a} - {alt-z}
{ctrl-0} - {ctrl-9}
{esc} {tab} {bksp} {enter}
{space} {ins} {del}
{ctrl-esc} {ctrl-tab} {ctrl-bksp} {ctrl-enter}
{ctrl-space} {ctrl-ins} {ctrl-del}
{ctrl-prtscr}

Beachten Sie, daß {ctrl-prtscr} die einzige Möglichkeit einers Bildschirmausdrucks über den Tastaturpuffer darstellt, die das Bios durchläßt.

{alt-esc} {alt-tab} {alt-bksp} {alt-enter}{alt-space}
{alt-ins} {alt-del}

Cursor-Steuertasten auf der numerischen Tastatur

Auf der numerischen Tastatur finden Sie folgende Tasten:

{up} {down} {left} {right}
{home} {end} {pgup} {pgdn}

{ctrl-up} {ctrl-down} {ctrl-left} {ctrl-right}
{ctrl-home} {ctrl-end} {ctrl-pgup} {ctrl-pgdn}
{alt-up} {alt-down} {alt-left} {alt-right}
{alt-home} {alt-end} {alt-pgup} {alt-pgdn}

Erweiterte Cursor-Steuertasten

Diese Tasten entsprechen den gesonderten Grautasten, die auf den erweiterten 101/102-er Tastaturen zu finden sind. Das Sternchen (*) verweist auf diese Grautasten:

{*ins} {*del} {*end} {*pgup} {*pgdn} {*home}
{*up} {*down} {*right} {*left}
{ctrl-*ins} {ctrl-*del} {ctrl-*end} {ctrl-*pgup}
{ctrl-*pgdn}{ctrl-*home} {ctrl-*up} {ctrl-*down}
{ctrl-*right} {ctrl-*left}
{alt-*ins} {alt-*del} {alt-*end} {alt-*pgup}
{alt-*pgdn}{alt-*home} {alt-*up} {alt-*down}
{alt-*right} {alt-*left}

Tasten der numerischen Tastatur

Wenn Sie normalerweise eine Zahl (0-9) als Teil eines eingebetteten Strings angeben, dann wird der Tastencode der numerischen Tasten oberhalb der normalen Tastatur generiert . Wenn Sie die Tastencodes der numerischen Tastatur im Puffer ablegen müssen, dann sollten Sie folgende Zeichenfolgen verwenden (das #-Zeichen verweist darauf, daß der Tastencode aus der numerischen Tastatur stammt):

{#0} {#1} {#2} {#3} {#4} {#5}
{#6} {#7} {#8} {#9} {#.}
{#/} {#*} {#-} {#+} {#enter}
{ctrl-#0} {ctrl-#1} {ctrl-#2} {ctrl-#3} {ctrl-#4}
{ctrl-#5}{ctrl-#6} {ctrl-#7} {ctrl-#8} {ctrl-#9} {ctrl-#.}
{ctrl-#/} {ctrl-#*} {ctrl-#-} {ctrl-#+} {ctrl-#enter}
{alt-#0} {alt-#1} {alt-#2} {alt-#3} {alt-#4} {alt-#5}
{alt-#6} {alt-#7} {alt-#8} {alt-#9} {alt-#.}
{alt-#/} {alt-#*} {alt-#-} {alt-#+} {alt-#enter}

Sie können die Strings und nicht-ASCII-Tastenkombinationen durch Leerzeichen voneinander trennen.

eg:

```
x = SWPKEYBRD(" <tab> 'Teste...' <enter>" )
```

Internationale Tastaturtreiber

An den untergeordneten Prozeß übergebene Tastendrücke werden direkt vom Swapkern durch Übergabe eines möglicherweise geladenen internationalen Tastaturtreibers übergeben. Dadurch scheinen die Tasten von einer US-Tastatur zu stammen. Der internationale Tastatur-Handler übernimmt wieder die Steuerung, sobald SWPKEYBRD() alle Tastendrücke in den Puffer abgelegt hat.

Umwandlung der Tastencodes

Es gibt zwei verschiedene Typen des BIOS-Aufrufs, die ein Programm bei dem Einlesen von der Tastatur verwenden kann: der eine Typ für die ‘Standard-’Tasten und der andere für

die ‘erweiterten Tasten’. SWPKEYBRD() liefert automatisch den richtigen Tastencode für den verwendeten BIOS-Aufruf.

Löschen des Tastaturpuffers

Standardmäßig gaukelt SWPKEYBRD() dem untergeordneten Programm vor, daß der Tastaturpuffer zwischen jedem Tastendruck gelöscht wird. Dadurch kann die Tastenablagefunktion auch mit Programmen verwendet werden, die den Tastaturpuffer regelmäßig löschen (und daher nachfolgend abgelegte Tastencodes verlieren würden). Dieses Verhalten kann mit Hilfe der Funktion SWPKEYCLR() verändert werden.

Wie funktioniert SWPKEYBRD()?

Der an SWPKEYBRD() übergebene String wird sofort in die entsprechende Tastencodefolge umgewandelt. Wenn SWPRUNCMD() aufgerufen wird, um das untergeordnete Programm auszuführen, übernimmt der Swap-Kern (das ist der Teil von SWPRUNCMD(), der während der Auslagerung im Speicher verbleibt.), den BIOS-Tastatur-Interrupt und füllt die Tastencodes in den Tastaturpuffer des untergeordneten Programms als Reaktion auf die vom untergeordneten Programm (oder DOS) gestarteten BIOS-Aufrufe, als würden die Tastencodes direkt von der Tastatur kommen. Wurden alle Tasten abgelegt, dann gestattet der Kern wieder die normale Tastatureingabe für das untergeordnete Programm.

Die meisten Programme, die entweder die DOS- oder die BIOS-Funktionsaufrufe zum Lesen der Tastatureingabe verwenden, können über SWPKEYBRD() gesteuert werden.

Erstellung einer Textdatei über einen bekannten Texteditor:

Dieses Beispiel legt die erste Zeile, gefolgt von einem {Enter}-Tastencode, im Puffer ab, danach die zweite Zeile, wiederum gefolgt von einem {enter}. Als nächstes wird die Datei über die Eingabe der Tastencodes {alt-f} (Menü File (Datei)) und 's' (für save (speichern)) abgespeichert. Abschließend wird das Editor-Programm über {alt-x} beendet.

```
keystr := " 'Mary hatte ein kleines Lamm,' {enter} "
keystr := keystr + ' "Sein Fell war..." {enter}'
keystr := keystr + "{alt-f}'s'{alt-x}"
swpkeybrd(keystr)
swpruncmd("EDIT MARY",0,"","")
```

Es folgt die Eingabe der Daten in ein (fiktives) Spreadsheet-Programm sowie die Neuberechnung, Abspeicherung und das Beenden des Programms:

```
keystr = "{down}{down}{right}{right}"
keystr = keystr + " '2.75'{enter}"
keystr = keystr + " '!' {enter} "
keystr = keystr + " '/s' {enter} '/x' {enter} "
SWPKEYBRD(keystr)
SWPRUNCMD("SS TEST",0,"","")
```

Beispiel:

```
// Tastatureingaben für den DIR Befehl eingeben und dann nach DOS
// swappen und ausführen:
SWPKEYBRD("'DIR'{enter}")
SWPRUNCMD("",0,"","")
```

Siehe auch: SWPKEYCLR()

SWPKEYCLR()

Funktion: Verändert das Verhalten von SWPKEYBRD()

Syntax: `niWert = SWPKEYCLR(niClearCount)`

Parameter: niClearCount Anzahl der Löschungen des Tastaturpuffers

Rückgabe: Die vorhergehende Einstellung.

Bemerkung: Diese Funktion gibt an, wie oft SWPKEYBRD() den Tastaturpuffer zwischen der Übertragung von Tastaturcodes an den untergeordneten Prozeß löscht.

Normalerweise gibt SWPKEYBRD() dem untergeordneten Programm an, daß der Tastaturpuffer nach jedem abgelegten Tastencode leer ist. Das verhindert, daß Tastendrücke verlorengehen, wenn das untergeordnete Programm versucht, den Tastaturpuffer selbst zu löschen.

In Multitasking-Systemen, wie DesqView, liest das Betriebssystem jedoch selbst die Tastatur, um Wartezeiten zu erkennen, und 'stiehlt' damit das Tastaturpuffer Löschsignal,das der Swap-Kern an das untergeordnete Programm zu senden versucht. Daher sendet der Swap-Kern nicht ein, sondern 5 Tastaturlöschsignale zwischen den einzelnen Tastencodes, um abzusichern, daß mindestens eines von ihnen bei dem untergeordneten Programm ankommt.

Wenn es so aussieht, daß Tastencodes verloren gehen, dann ist es unter Umständen hilfreich, diesen Wert einfach zu erhöhen.

Beispiel:
```
SWPKEYCLR(10) // 10mal Löschen zwischen jedem Tastendruck
```

Siehe auch: SWPKEYBRD() / SWPKEYBRDBAS()

SWPNOBOOT()

Funktion: Verhindern eines Reboot durch den Anwender

Syntax: `lWert = SWPNOBOOT(lWert)`

Parameter: lWert logischer Wert für ein/aus

Rückgabe: Die vorherige Einstellung

Standard: Ausgeschaltet

Bemerkung: Diese Funktion dient dem Ein-Ausschalten der Überwachung der <STRG><ALT><ENTF>-Tastatursequenz, solange das Swap-Kernel sich im Speicher befindet. Dieses verbietet dem Anwender den Rechner zu rebooten, solange das Programm ausgelagert ist. Beachten Sie bitte, daß dieser Mechanismus nicht unter DesqView oder ähnlichen Umgebungen funktioniert, die sich in den Tastatur-Vektor hängen und nach dieser Sequenz Ausschau halten, da diese mit der Bearbeitungsreihenfolge eher zum Zug kommen, als der Swap-Kernel.

Beispiel:
```
SWPNOBOOT(.T.)   // Warm-Boot mit Ctrl Alt Del
 // ausschalten während des Swappens
```

SWPRUNCMD() / SWPRUNCMDBAS()

Funktion: Einen Kind-Prozeß ausführen

Syntax:
```
lWert = SWPRUNCMD(cBefehl, nuMem, cRunPfad, cTempPfad)
```

Parameter:

cBefehl	Befehl,der ausgeführt werden soll
nuMem	Speicher in KB, der verwendet werden soll
cRunPfad	Einstellung für Laufwerk und Verzeichnis
cTempPfad	Pfad für temporäre Dateien

Rückgabe: TRUE, falls der Kommando-Interpreter ausgeführt wurde.
FALSE, falls ein Fehler aufgetreten ist

Bemerkung: Diese Funktion führt einen Kind-Prozeß aus, nachdem der von dem Eltern-Prozeß belegte konventionelle Hauptspeicher freigegeben wurde. Der von dem Elternprozeß benutzte nichtkonventionelle Speicher bleibt belegt.

Die Parameter von SWPRUNCMD() schlüsseln sich wie folgt auf:

cBefehl: Der Name des Programms, der Batch-Datei oder des DOS-internen Befehls, der ausgeführt werden soll, gefolgt von etwaigen, benötigten Parametern. Ein Leerstring sorgt dafür, daß nur der Kommando-Interpreter aufgerufen wird, welcher den DOS-Prompt liefert.

nuMem: Die Menge Speicher in Kilobytes, die freizugeben sind, bevor der Kind-Prozeß ausgeführt wird. Ist dieser Parameter 0, wird soviel Speicher wie möglich freigegeben.

cRunPfad: Laufwerk und Verzeichnis, das vor dem Start des Kind-Prozesses als aktuell eingestellt werden soll.

cTempPfad: Das Laufwerk samt Pfad, in dem die Swap-Funktion ein Swap-File anlegen soll, das ein Abbild (oder Teile davon) des Eltern-Prozesses enthält. Beachten Sie, daß soweit möglich XMS und EMS für die Unterbringung der Swapdaten verwandt werden, sofern dies nicht durch die Verwendung von SWPUSEEMS() und SWPUSEXMS unterbunden wurde.

Beachten Sie, daß dieser Parameter zur Laufzeit überschrieben werden kann, indem die SWAP- oder BLINKER-Umgebungsvariable verwendet wird. Ein Beispiel:

```
SET SWAP=F:\TEMP;
```

oder

```
SET BLINKER=/SDF:\TEMP
```

was dazu führt, daß ein temporärer Plattenfile auf F:\TEMP angelegt wird, ganz gleich, welcher Pfad an SWPRUNCMD() übergeben wurde.

Die Swap-Funktion ruft immer den Kommando-Interpreter auf, der durch die COMSPEC-Umgebungsvariable gesetzt worden ist, welcher seinerseits dann wieder den Befehl ausführt, der durch cKommando gesetzt wurde. Dies erlaubt dem Eltern-Prozeß jeglichen internen oder externen DOS-Befehl, eine Batch-Datei oder irgendein Programm auszuführen, das vom DOS-Prompt gestartet werden kann. Der Rückgabewert des Kind-Prozesses kann mit der Funktion SWPERRLEV() ermittelt werden.

Der Rückgabewert ist ein logischer (boolscher) Wert, der aussagt, ob die Aktion von

SWPRUNCMD() erfolgreich war, oder nicht. Wurde SWPRUNCMD() nicht erfolgreich ausgeführt, kann mittels SWPERRMAJ() und SWPERRMIN() die Art des Fehlers bestimmt werden.

Beispiel:

```
success = SWPRUNCMD("",0,"","")
```

DOS ausführen mit soviel Speicher wie möglich, die Swapdatei wird, falls erforderlich, im aktuellen Verzeichnis gespeichert.

```
success = SWPRUNCMD("WIN",0,"C:\WINDOWS","F:\")
```

Dieser Befehl ruft MS-Windows auf, nachdem soviel Speicher wie möglich freigegeben wurde. Die Swap-Funktion wechselt das Laufwerk und Verzeichnis nach C:\WINDOWS, bevor Windows gestartet wird. Sollte eine temporäre Plattendatei benötigt werden, so wird diese auf F:\ erstellt

```
success = SWPRUNCMD("DIR /W",100,"","")
IF (!success)
  ? "Swap fehlgeschlagen"
  majerr = SWPERRMAJ()
  minerr = SWPERRMIN()
  ? "Fehlercode ", majerr, minerr
ELSE
  ? "Swap erfolgreich"
  ? "Fehler :",SWPERRLEV()
ENDIF
```

Der DOS-Befehl Dir wird ausgeführt, nachdem 100 KB Speicher freigegeben wurden. Die Swap-Datei wird im aktuellen Verzeichnis angelegt, sollte Sie benötigt werden. Bei Rückkehr überprüfe den Rückgabewert. Falls der Swap fehlgeschlagen ist, zeige Haupt- und Unter-Fehlercodes an, falls er erfolgreich war, zeige den Rückgabewert des Unterprozesses an.

```
success = SWPRUNCMD("RRUN ABC",0,"","")
IF (!success)
  ? "Swap fehlgeschlagen"
  majerr = SWPERRMAJ()
  minerr = SWPERRMIN()
  ? "Fehlercode ", majerr, minerr
ELSE
  ? "Swap erfolgreich"
  ? "Fehler :",SWPERRLEV()
ENDIF
```

Rufe den R&R Reportwriter auf, Prüfe auf Fehler bei der Rückkehr.

Siehe auch:

SWPERRLEV()
SWPERRMAJ()
SWPERRMIN()
Swap-Laufzeitfehlermeldungen (1220)
Parameterwert außerhalb des zulässigen Bereichs

SWPSETENV() / SWPSETENVBAS()

Funktion: Setzt die Umgebungsvariablen für den Kind-Prozeß

Syntax:
```
lsuccess = SWPSETENV(cString)
```

Parameter: cString String mit Umgebungsvariablen

Rückgabe: True, wenn cString kürzer als 512 Bytes ist, sonst False

Bemerkung: SWPSETENV() ermöglicht dem übergeordneten Programm, Umgebungsstrings in der Umgebung des untergeordneten Programms hinzuzufügen, zu ändern oder zu löschen. Der Parameter der Funktion SWPSETENV() stellt einen String mit all den Variablen dar, die gesetzt werden sollen. Die Variablen werden durch das Begrenzerzeichen CHR(255) voneinander getrennt. Am Ende des Strings ist kein CHR(255) erforderlich. Wenn im String selbst ein CHR(255) benötigt wird, dann geben Sie zwei hintereinander an.

Beispiel:
```
str := "PROMPT=<Type Exit to return><$p$g" +;
          CHR(255)+"SWAPPED=TRUE"
SWPSETENV(str)
```

Dieses Beispiel verändert den Prompt und fügt (oder ändert) die Umgebungsvariable 'SWAPPED' hinzu.

```
str := "CLIPPER="+CHR(255)+"BLINKER=/OP1"
SWPSETENV(str)
```

Dieses Beispiel LÖSCHT die Umgebungsvariable 'CLIPPER' und ändert / fügt die Variable 'BLINKER' hinzu.

Beachten Sie, daß diese Umgebungsänderungen nur während einer Auslagerung durchgeführt werden können und sich nur auf die Umgebung des untergeordneten Programms auswirken und NICHT auf die Kopie des übergeordneten Programms.

Durch die Verwendung dieser Funktion wird der Kern im Speicher um den Umfang der neuen Umgebung vergrößert.

Siehe auch: SWPADDENV()

SWPSETPID() / SWPSETPIDBAS()

Funktion: Setzen einer Programm-ID für das aktuell ausgeführte Programm

Syntax: `lWert = SWPSETPID(cIdString)`

Parameter: cIdString zu setzende Programm-ID

Rückgabe: TRUE, wenn die ID gesetzt werden konnte

Bemerkung: Setzen der Programm-ID des aktuellen Programms auf cldString. Die maximale Länge von cldString ist 31 Zeichen. Ist die Stringlänge größer als 31, liefert die Funktion immer FALSE und die Programm-ID wird nicht gesetzt.

Eine Programm-ID kann immer gesetzt werden, sogar unabhängig davon, ob es die gleiche ID für ein höherstufiges-Programm schon gibt. Ein Kind-Prozeß kann ermitteln, ob eine ID gesetzt wurde mittels der Funktion SWPGETPID()

Beispiel:

```
*
* Setzen der Programm-ID auf "Test Programm"
*
IdStr = "Test Programm"
SWPSETPID(IdStr)
```

Siehe auch: SWPADDSTR() / SWPADDSTRBAS()
SWPGETPID() / SWPGETPIDBAS()
SWPGETSTR() / SWPGETSTRBAS()
SWPSETSTR() / SWPSETSTRBAS()

SWPSETSTR() / SWPSETSTRBAS()

Funktion: Definiert einen String, der an den übergeordneten Prozeß zurückgegeben werden soll.

Syntax: `lWert = SWPSETSTR(cPID, cString)`

Parameter: cPID ID des übergeordneten Prozesses

cString Sring, der an das übergeordnete Programm zurückgegeben wird

Rückgabe: Ein logischer Wert

Bemerkung: Über SWPSETSTR() kann ein String von bis zu 127 Bytes von dem untergeordneten Prozeß an den übergeordneten übergeben werden. Der Rückgabestring darf keine NULL-Zeichen enthalten. Das erste auftretende NULL-Zeichen wird als Stringendezeichen interpretiert.

Das übergeordnete Programm, an das der String gesendet wird, wird über seine Programm-ID identifiziert (Diese ID wurde im übergeordneten Programmm über die Funktion SWPSETPID() gesetzt). Empfangen werden kann der String vom übergeordneten Programm über die Funktion SWPGETSTR().

Beispiel: `success := SWPSETSTR("TEST","OK")`

Liefert den String "OK" an das Programm mit der Programm-ID "TEST".

Der Rückgabewert ist ein logischer Wert, der angibt, ob der Prozeß erfolgreich abgelaufen ist oder nicht. Für Fehler gibt es drei mögliche Ursachen:

- 1 - Der Kern des angegebenen übergeordneten Programms befindet sich nicht im Speicher.
- 2 - Der Rückgabestring ist länger als 127 Bytes.
- 3 - Das übergeordnete Programm wurde mit einer Blinker-Version gelinkt, die alter als 2.10 ist.

Siehe auch: SWPADDSTR() / SWPADDSTRBAS()
SWPGETPID() / SWPGETPIDBAS()
SWPGETSTR() / SWPGETSTRBAS()
SWPSETPID() / SWPSETPIDBAS()

SWPUSEEMS()

Funktion: Benutzung von EMS-Speicher durch das Swap-System ein- oder ausschalten.

Syntax:
```
lWert = SWPUSEEMS(lWert)
```

Parameter: lWert Logischer Wert für ein/aus

Rückgabe: Die vorherige Einstellung

Standard: Eingeschaltet

Bemerkung: Diese Funktion ermöglicht das Ein-/Ausschalten der Verwendung von EMS-Speicher für das Swap-System. Der Rückgabewert entspricht der vorherigen Einstellung. Ist die Verwendung von EMS eingeschaltet, kann das Programmabbild des übergeordneten Programms wesentlich schneller gespeichert und wieder geladen werden, als wenn es auf Platte abgelegt worden wäre.

Beispiel:
```
SWPUSEEMS(.F.)      // Verwendung von EMS für Swappen ausschalten
```

Hinweis: Beachten Sie bitte, daß, sollte sowohl XMS als auch EMS vorhanden sein, zunächst XMS-Speicher verwendet wird.

Beachten Sie auch, daß die Einstellung der Umgebungsvariablen BLINKER /SEc die über diese Funktion angegebenen Einstellungen überschreibt.

Siehe auch: SWPUSEUMB()
SWPUSEXMS()

SWPUSEUMB()

Funktion: Benutzung von UMB-Speicher durch das Swap-System ein- oder ausschalten.

Syntax: `lWert = SWPUSEUMB(lWert)`

Parameter: lWert Logischer Wert für ein/aus

Rückgabe: Die vorherige Einstellung

Standard: Eingeschaltet

Bemerkung: Diese Funktion schaltet die Verwendung von UMBs (Upper Memory Blocks) für den residenten Kern ein oder aus. Falls UMBs vorhanden und ihre Verwendung eingeschaltet ist, verlegt sich der residente Kern, der während der Ausführung des Unterprozesses im Speicher bleibt, in ein UMB und macht dadurch etwas mehr Speicher frei.

Beispiel:
```
SWPUSEUMB(.F.)      // Verwendung von UMBs für Swappen ausschalten
```

Hinweis: Beachten Sie auch, daß die Einstellung der Umgebungsvariablen BLINKER /SUc diese Funktion überschreibt.

Siehe auch: SWPUSEEMS()
SWPUSEXMS()

SWPUSEXMS()

Funktion: Benutzung von XMS-Speicher durch das Swap-System ein- oder ausschalten.

Syntax: `lWert = SWPUSEXMS(lWert)`

Parameter: lWert Logischer Wert für ein/aus

Rückgabe: Die vorherige Einstellung

Standard: Eingeschaltet

Bemerkung: Diese Funktion ermöglicht das Ein-Ausschalten der Verwendung von XMS-Speicher für das Swap-System. Der Rückgabewert entspricht der vorherigen Einstellung. Ist die Verwendung von XMS eingeschaltet, kann das Eltern-Image wesentlich schneller gespeichert und wieder geladen werder, als wenn es auf Platte abgelegt worden wäre.

Beispiel:
```
SWPUSEXMS(.F.)      // Verwendung von XMS für Swappen ausschalten
```

Hinweis: Beachten Sie bitte, daß, sollte sowohl XMS als auch EMS vorhanden sein, zunächst XMS-Speicher verwendet wird. Beachten Sie auch, daß die Einstellung der Umgebungsvariable BLINKER /SXc diese Funktion überschreibt.

Siehe auch: SWPUSEEMS()
SWPUSEUMB()

SWPVIDMDE()

Funktion: Sichern des Video-Modus und der Video-Page nach einer Swap-Aktion ein- oder ausschalten.

Syntax: `lWert = SWPVIDMDE(lWert)`

Parameter: lWert Logischer Wert für ein/aus

Rückgabe: Die vorhherige Einstellung

Standard: Ausgeschaltet

Bemerkung: Ist diese Option aktiviert, so sichert das Swap-Kernel den Video-Modus und die aktuelle Bildschirmseite bevor die Swap-Aktion gestartet wird. Ebenso werden diese gesicherten Werte nach den Swap wieder zurückgesetzt. Beachten Sie aber, daß der Inhalt des Video-Speichers nicht gesichert wird.

Beispiel:
```
SWPVIDMDE(.T.)      // Videomode beim Swappen speichern
```

Fehler und Fehlermeldungen

Fehlermeldungen Überblick

Lösungsansätze für die Fehlersuche

Linkzeit-Warnungen (10xx)

Schwerwiegende Linkzeit-Fehler (11xx)

DOS-Real Mode-Laufzeitfehler (12xx)

Swap-Laufzeitfehlermeldungen (1220)

DOS-Protected Mode-Laufzeitfehler (13xx)

Andere häufige Fehler

Fehlermeldungen Überblick

Die folgenden Abschnitte enthalten eine komplette Liste aller Blinker-Fehlermeldungen. Zu jeder dieser Meldungen wird eine kurze Erläuterung des Fehlers und den möglichen Gründen seines Auftretens gegeben.

Die Fehler sind in folgende Klassen eingeteilt:

- 10xx - Linkzeit-Warnungen
- 11xx - fatale Linkzeit-Fehler
- 12xx - DOS-Real Mode-Laufzeit-Fehler
- 1220 - Swap-Laufzeit-Fehler
- 13xx - DOS-Protected Mode-Laufzeit-Fehler
- Andere allgemeine Fehler

DOS Errorlevel

Real Mode Error-Handler von Blinker

Hinweis für CA-Clipper

Fehlermeldungen Überblick

DOS Errorlevel

Linkzeit-Warnungen dienen ausschließlich der Information und setzen den DOS-Errorlevel nicht. So ist gewährleistet, daß sowohl der Blinker selbst, als auch jegliches MAKE-Utility weiterhin ihre Arbeit verrichten. Die Ausgabe dieser Warnungen kann mit Hilfe des Befehls BLINKER MESSAGE NOWARNING unterdrückt werden.

Linkzeit-Fehler und Laufzeit-Fehler sind schon ernsterer Natur. Sie haben eine sofortige Beendigung des Link-vorganges zur folge und setzen den DOS-Errorlevel auf 1.

Fehlermeldungen Überblick

Real Mode Error-Handler von Blinker

Sollte während der Laufzeit eines mit Blinker erzeugten Real Mode Programms ein Fehler auftreten, dann übergibt es den entsprechenden Code an das Reaaal Mode Fehlerbehandlungs-Modul. Diese Routine liefert den Text der Meldung, die am Bildschirm anzuzeigen ist. Blinker fügt diese Fehlerbehandlungs-routine während des Linkes automatisch in die .EXE-Datei ein.

Diese Fehlerbehandlungsroutinen sind als Programm-quelltexte verschiedener Programmiersprachen im Lieferumfang des Blinker enthalten. Dadurch können die Fehlermeldungen individuell angepaßt werden..

Sollten Sie die Blinker-Fehlerbehandlungsroutine modifizieren, müssen Sie diese neu kompilieren und als FILE mit in Ihre eigene Applikation linken. Falls sie nicht modifiziert werden soll, besteht keine Notwendigkeit zum Neukompilieren, da sie automatisch durch Blinker in die .EXE-Datei eingebunden wird.

Lassen Sie bitte Vorsicht walten, falls Sie vorhaben die Fehlerbehandlungsroutinen in ein Overlay auszulagern. Sollte in einem solchen Fall ein Fehler 1201:"Overlaydatei nicht im angegebenen Pfad" auftreten, würde dies zu einer endlosen Rekursion führen. D.h., daß beim Auftreten dieses Fehlers nach dem Overlay gesucht wird, daß den Error-Handler enthält. Da dieser aber nicht auffindbar ist, wird erneut danach gesucht, und so weiter und so fort.

Der Entwickler kann solche Fehler auch an anderer Stelle abfangen um das Problem zu umgehen, oder er stellt einfach sicher, daß das entsprechende Overlay auffindbar ist indem er es z.B. als internes Overlay auslegt.

Fehlermeldungen Überblick

Hinweis für CA-Clipper

Der Error-Handler liefert den Fehlertext zu dem durch die Funktion BLIERRNUM() zurückgegebenen Fehlercode, oder in einigen Fällen auch durch BLIERRPRM(), um an den Namen des fehlerhaften Elements zu kommen. Im Falle von CA-Clipper Sommer ’87 wird im Anschluß daran, die Funktion MISC_ERROR mit einem Prozedurnamen von BLINKER aufgerufen, einer Zeilennummer, welcher dem Fehlercode entspricht und die dazu passende Fehlermeldung. Im Falle von CA-Clipper 5.x wird ein neues Fehler-Objekt erzeugt, welches als Argument dem Standard-Fehlerblock übermittelt wird.

Lösungsansätze für die Fehlersuche

Gewöhnlich sind Linkzeit-Fehler gut zu identifizieren und der Ansatz zur Problemlösung wird durch die individuelle Beschreibung der Fehlermeldungen im folgenden Abschnitt gegeben. Es stehen folgende zusätzliche Möglichkeiten zum Ausschließen von Linkzeit-Problemen zur Verfügung:

- Der Linkbefehl VERBOSE zur Anzeige, welche Aufgabe Blinker gerade durchgeführt hat, als der Fehler auftrat.
- Der Befehl MAP zur Anzeige, wie eine .EXE-Datei aufgebaut ist.
- Der Befehl ECHO zur Ausgabe eigener Meldungen während der Abarbeitung der Link-Skriptdatei.

Auf der anderen Seite können Laufzeitfehler, die durch falsches Linken hervorgerufen werden, wesentlich komplizierter und infolgedessen auch schwerer zu diagnostizieren und zu beseitigen sein.

Werden Fehler festgestellt, die sich nicht aufgrund des Codes oder anderer Faktoren der Umgebung nachvollziehen lassen, sollten folgende Schritte unternommen werden, um die Rolle des Linkers bei dem Problem herauszufinden.

1. Gewährleisten Sie einen genügend großen Stapelspeicher mittels des STACK nnnn oder BLINKER PROCEDURE DEPTH nn Befehls.

2. Suchen Sie nach doppelten Modulen mittels des BLINKER MESSAGE DUPLICATES-Befehls. Sehen Sie inder Online-Hilfe nach der Beschreibung dieses Befehls. Gewöhnlich kann diese Situation dadurch gelöst werden, daß die Reihenfolge der Bibliotheken umgestellt wird. Verwenden Sie hierzu ggf. den MODULE FROM-Befehl.

3. Versuchen Sie, das Problem mit einem kleinen Testprogramm zu erzeugen. Oder versuchen Sie, einen kleinen Teil des Programms zu isolieren, indem Sie nach und nach Programmteile entfernen, die nicht aufgerufen werden müssen.

4. Testen Sie das Original- und das kleinere Testprogramm mit einem anderen Linker. Sollte das Problem mit dem anderen Linker nicht auftreten, so verwenden Sie nocheinmel den Blinker mit den gleichen Modulen und der gleichen Reihenfolge der Bibliotheken.

5. Benachrichtigen Sie den Technischen Support von Blinker über die Webseite www.blinkinc.com . Der Bereich ‘Technical Support’ enthält die neuesten Updates und technische Hinweise. Bitte sehen Sie diese erst durch, bevor Sie eine Email-Nachricht senden oder unseren technischen Support anrufen. Suchen Sie bitte auch in der Dokumentation nach weiteren Informationen zu Ihrem Problem. Sollten Sie

den telefonischen Support dennoch benötigen, bitten wir Sie, Ihre Blinker-Seriennummer, Link-Skript-Dateien und den genauen Text der Fehlermeldung bereitzuhalten.

Linkzeit-Warnungen (10xx)

Fehlermeldungen mit den Nummern 1001 bis 1099 sind Warnungen, die während der Linkzeit erzeugt werden.

1001: Ungültiger Befehlsparameter in der Linkdatei in‘<Befehl>’ wird ignoriert

Diese Nachricht zeigt an, daß der angegebene Linkbefehl ignoriert wurde, da er einen ungültigen Parameterwert enthält.

Sehen Sie bitte in die Online-Hilfe, Abschnitt Blinker-Befehle, um Details zu den benötigten Parametern des angezeigten Befehls zu erhalten.

1002: Zu verarbeitende Datei <Dateiname> doppelt angegeben, ignoriert

Diese Nachricht zeigt an, daß die Eingabe-Datei mehr als einmal innerhalb eines FILE- oder LIBRARY-Kommandos aufgeführt ist. Diese doppelte Datei wird ignoriert.

1003: Warnung - kein Stacksegment in .EXE-Datei

Diese Nachricht zeigt an, daß das .EXE ohne ein Stack-Segment erzeugt wurde, was einen Programm-Abbruch zur Laufzeit zur Folge haben könnte, sofern es nicht dahingehend ausgelegt ist, sich während der Laufzeit seinen eigenen Stack zu erzeugen.

1004: <Dateiname(Modulname)> : ‘<Symbol>’ ist dupliziert in <Dateiname(Modulname)>

Diese Meldung besagt, daß das Symbol <Smbol> sowohl im ersten Modul als auch im zweiten Moul als public definiert ist.

Dies passiert, wenn zwei Symbole in unterschiedlichen .OBJ oder .LIB Dateien den gleichen Namen haben. Das zuerstgefundene wird in das Programm gelinkt, das andere wird ignoriert.

1004: DEFINE Befehl : '<Symbol>' dupliziert in <Dateiname(Modulname)>

Diese Meldung besagt, daß das Symbol <Symbol> , das in einem DEFINE-Befehl als public definiert wird, bereits in einem anderen Modul als public deklariert ist.

Dies passiert, wenn eine .OBJ oder .LIB Datei ein Symbol mit dem gleichen Namen wie das Symbol in der DEFINE Anweisung enthält. Das Symbol in der .OBJ oder .LIB Datei wird dann ignoriert.

1005: Die vorgegebene Overlay-Klasse ‘<class>’ enthält das Wort ‘DATA’

Diese Nachricht zeigt an, daß die Standard-Overlay-Klasse, welche sich aus dem ersten Segment des ersten Objekts ergibt, das Wort ‘DATA’ enthält. Blinker kann nur Code-Segmente in Overlays legen, daher erscheint diese Meldung als Warnung, falls eine nicht-standard .OBJ-Datei als erste .OBJ-Datei gelinkt wird.

Ist der angezeigte Klassen-Name keine Code-Klasse, überprüfen Sie bitte, ob das erste angegebene Objekt ein gültiges Code-Segment als erstes Segment enthält, oder setzen Sie eine Ihrer eigenen .OBJ-Dateien als erste in das Linkskript.

Ist der angezeigte Klassenname ein korrekter Name einer Codeklasse, die in ein Overlay abgelegt werden soll, dann können Sie diese Warnung durch Einfügen des Befehls OVERLAY CODE, <class> in die Linkdatei überschreiben.

1006: <Dateiname>: Füllzeichen abgeschnitten, um in 64 KB zu passen

Diese Nachricht ist eine Warnung, daß die Prozedur in der angegebenen Datei fast die 64KB-Grenze erreicht, was wiederum bedeutet, daß nicht genügend Freiraum für das inkrementelle Linken zur Verfügung steht.

Blinker begrenzt diesen Leerraum automatisch so, daß das 64KB Limit nicht überschritten wird. Trotzdem ist es für das Overlaying effektiver und gleichfalls eine gute Programmierpraxis, Prozeduren möglichst klein zu halten. Typischerweise sollten es nicht mehr als einige 100 Zeilen Code sein.

1007: Warnung - Das Program enthält keine Startadresse

Diese Meldung zeigt an, daß während des Linkens kein Modul mit einenm Datensatz gefunden wurde, der auf den Startpunkt des Programms zeigt. Ein Modul dieses Typs befindet sich gewöhnlich in der Standardsprachbibliothek des Compilers.

1008: <Dateiname> : Ungültige Datei für inkrementelles Linken

Diese Nachricht zeigt an, daß das vorherige inkrementelle Linken erfolglos war und eine ungültige inkrementelle Linkdatei (.BIF) übrig blieb. Möglich ist auch, daß die Datei, die mit dem Befehl BLINKER INCREMENTAL FILE angegeben wurde, keine gültige inkrementelle Linkdatei war. Es wird stattdessen ein kompletter Linklauf ausgeführt, bei dem sowohl die .EXE-Datei, als auch die .BIF-Linkdatei neu aufgebaut wird.

1009: <Dateiname>: Datei kann nicht inkrementell gelinkt werden

Diese Nachricht gilt nur für CA-Clipper-Applikationen und bedeutet, daß während des Linkvorgangs durch Blinker erkannt wurde, daß eine nicht-CA-Clipper-Datei geändert ist. Inkrementelles Linken von nicht-CA-Clipper-Dateien wird zur Zeit noch nicht unterstützt. Ein kompletter Linklauf wird statt dessen ausgeführt.

1010: <Dateiname>: kein Platz für inkrementelles Linken verfügbar

Diese Nachricht zeigt an, daß die angegebene Datei zu groß wurde, um in den beim letzten vollen Linken reservierten Platz zu passen. Ein voller Linklauf wird durchgeführt, um die .EXE-Datei und die zugehörige inkrementelle Linkdatei neu zu erzeugen.

Sollten Sie diese Nachricht häufiger erhalten, kann durch den Befehl BLINKER INCREMENTAL PAD <padgroesse> mehr Platz für inkrementelles Linken bereitgehalten werden (siehe Beschreibung in der Online-Hilfe). Eine erweiterte Pad-Größe führt zu einem besseren Verhältnis zwischen inkrementellen und vollen Linkläufen. Gleichzeitig aber steigt auch die Größe der .EXE-Datei und die Speicheranforderungen während derAusführung.

1011: ‘<symbol>’ referenziert zwei verschiedene Symbole

Diese Meldung teilt Ihnen mit, daß das angegebene Symbol mehr als einen Symbolnamen referenziert.

Die zweite Referenz für dieses Symbol wird ignoriert.

1012: Segmente in GROUP <Groupname> haben widersprüchliche Attribute

Diese Meldung zeigt an, daß über die SEGMENTS-Anweisungen in der Moduldefinitionsdatei für Segmente der angegebenen Gruppe unterschiedliche Attributeinstellungen angegeben wurden. Segmente

der gleiche Gruppe werden als ein Segment gespeichert und müssen daher allesamt die gleichen Attribute besitzen.

Die letzte in der Definitionsdatei gefundene Einstellung wird verwendet.

1013: Ungültiger Befehlsparameter in Moduldefinitionsdatei in‘<Befehl>’ wird ignoriert

Diese Meldung zeigt an, daß der angegebene Moduldefinitionsbefehl ignoriert wurde, da er einen unzulässigen Parameterwert enthält.

Einzelheiten über die Parameter des angezeigten Befehls finden Sie im Abschnitt "Moduldefinitionsbefehle" der Online-Hilfe.

1014: Doppelter Befehl '<Befehl>' in Moduldefinitions-Datei wird ignoriert

Diese Nachricht gibt an, daß der angegebene Befehl ignoriert wurde, da bereits zuvor ein Befehl des gleichen Typs aufgetreten ist und nur ein Befehl erlaubt ist.

1015: Doppelte Exportdefinition für Symbol ‘<symbol>’ wird ignoriert

Diese Meldung besagt, daß eine doppelte Exportdefinition für das angegebene Symbol gefunden wurde und diese ignoriert wurde.

1016: Doppelte Importdefinition für Symbol wird ignoriert '<symbol>'

Diese Meldung weist darauf hin, daß eine doppelte Importdefinition für das angegebene Symbol gefunden wurde, die aber ignoriert wird.

1017: Doppelte Definition für Segment wird ignoriert ''

Diese Meldung weist darauf hin, daß eine doppelte Segmentdefinition für den angegebenen Segmentnamen gefunden wurde, die aber ignoriert wird.

1018: kann das Symbol '<symbol>' nicht exportieren

Diese Meldung weist darauf hin, daß eine Exportdefinition für das angegebene Symbol gefunden wurde, das Symbol aber im Programm nicht existiert, so daß die Exportdefinition ignoriert wurde.

Schwerwiegende Linkzeit-Fehler (11xx)

Fehlermeldungen 1101 bis 1199 bedeuten schwere Linkzeit-Fehler

1101: Abbruch durch Benutzer

Diese Nachricht zeigt an, daß der Anwender den Blinker während des Laufes mittels STRG-C oder STRG-Break abgebrochen hat.

1102: Es wurden keine zu verarbeitenden Dateien angegeben

Diese Nachricht zeigt an, daß kein FILE-Befehl auf der Blinker-Kommandozeile oder in einer der Skript-Dateien gefunden wurde. Blinker benötigt mindestens eine FILE-Anweisung, die angibt, welche Objektdateien zu linken sind.

1103: '.....' ist kein gültiger Linkbefehl

Diese Nachricht zeigt an, daß das angegebenen Kommando kein gültiger Linkbefehl für Blinker ist. Sehen Sie auch im Abschnitt "Blinker-Befehle" der Online-Hilfe nach, um eine Übersicht der gültigen Blinker-Befehle zu bekommen und wie diese eingesetzt werden)

1103: '.....' ist kein gültiger Moduldefinitionsbefehl

Diese Nachricht zeigt an, daß der angezeigte Befehl kein gültiger Moduldefinitionsbefehl ist. Eine Übersicht über die gültigen Moduldefinitionsbefehle und ihre Einsatzmöglichkeiten finden Sie in der Online-Hilfe im Abschnitt "Moduldefinitionsbefehle".

1104: unterschiedliche Anzahl von BEGINAREAs und ENDAREAs

Diese Nachricht zeigt an, daß eine unterschiedliche Anzahl von BEGINAREA und ENDAREA-Befehle in den Linkdatei(en) gefunden wurde, was eine ungültige Overlay-Stuktur zur Folge hat. Normalerweise ist nur die Angabe eines BEGINAREA und eines ENDAREA für Blinker's dynamisches Overlaying nötig. Sollte die Overlay-Stuktur von einem anderen Linker übernommen worden sein, so kann diese allein dadurch vereinfacht werden, daß die Struktur zu einem Overlay zusammengefaßt wird.

1105: maximale Verschachtelungstiefe der Skriptdatei überschritten

Diese Nachricht zeigt an, daß ein Skript-Dateiname (@<Dateiname>) in einem anderen Skript gefunden wurde, was die maximale Verschachtelungstiefe von 5 übersteigen würde.

Es ist möglich dieses Problem dadurch zu lösen, in dem man die Skriptdateien auf mehrere unterschiedliche Skript-Dateien aufteilt und diese dann auf der Kommandozeile übergibt.

Zum Beispiel:

```
BLINKER @skript1 @skript2
```

1106: nicht genug konventioneller Speicher zum Linken vorhanden

Diese Nachricht zeigt an, daß Blinker nicht genügend reellen DOS-Speicher belegen kann, um mit dem Linken fortzufahren.

Blinker verfügt zur Linkzeit über eine virtuelle Speicherverwaltung, welche EMS (3.2 oder höher), XMS (2.0 oder höher) oder Plattenplatz verwenden kann, wenn der konventionelle Speicher erschöpft ist. Trotzdem benötigt Blinker eine gewisse Menge konventionellen DOS Speichers, ohne die er seine Arbeit

nicht durchführen kann.

Um mehr Speicher zur Verfügung zu stellen, können Sie TSR-Programme entfernen, Blinker außerhalb eines MAKE-Programms starten oder die Anzahl der in der CONFIG.SYS angegebenen Buffer herabsetzen. Die Anzahl der SECTION INTO oder SEARCH-Befehle reduziert den Speicherplatz ebenfalls.

1107: Platte voll: Ausgabedatei kann nicht geschrieben werden

Diese Nachricht zeigt an, daß die Zielplatte voll ist. Es entstand ein Schreibfehler, der allerdings auch durch nicht vorhandene Schreibrechte in einem Netzwerk oder einen ungültigen Dateinamen hervorgerufen sein könnte. Sollte genügend Plattenplatz vorhanden sein, kann es allgemeine Probleme mit dieser Platte geben. Bitte probieren Sie aus, diverse Programme oder Dateien in diesen Pfad hineinzukopieren, um herauszufinden, ob das Problem genereller Natur ist. Ein Programm, das mit externen Overlays definiert wurde, benötigt zur Linklzeit wesentlich weniger temporären Plattenplatz, als ein Programm, das mit internen Overlays aufgebaut wurde.

1108: Stack-Überlauf während des Linkens

Diese Nachricht zeigt an, daß der Laufzeit-Stack für BLINKER.EXE übergelaufen ist und sollte unter normalen Umständen nicht erscheinen. Wird diese Nachricht angezeigt, installieren Sie Ihre Blinker-Software neu. Sollte der Fehler weiterhin auftreten, wenden Sie sich bitte an den technischen Support mit Angaben zur Seriennummer und Systemkonfiguration.

1109: Folgende Datei konnte nicht geöffnet werden: <Dateiname>

Diese Nachricht zeigt an, daß Blinker die angegebene Datei nicht finden konnte. Daher sollte der komplette Pfad für diese Datei angegeben werden. Ist für das Link-Skript keine Erweiterung angegeben, setzt Blinker die Endung .LNK voraus.

Wurde die Datei in der Linkdatei über einen FILE-Befehl angegeben, dann sollten Sie die Umgebungsvariable OBJ überprüfen (Dateien nach FILE haben die Standardendung .OBJ). Wurde sie über einen LIBRARY- oder SEARCH-Befehl angegeben, dann überprüfen Sie bitte die Umgebungsvariable LIB (Dateien nach LIB oder SEARCH haben die Standardendung .LIB).

Gehört die Datei zu den Blinker Protected Mode .OBJ oder .LIB Dateien, z.B. BLXxxxnn.LIB, dann schauen Sie bitte in das OBJ oder LIB Unterverzeichnis von Blinker. Falls die Datei nicht existiert, es dort aber andere Dateien mit ähnlichen Namen, aber mit anderen Erweiterungen gibt, dann starten Sie bitte das Blinker-Utility BLIBLD.EXE im Hauptverzeichnis von Blinker. Damit werden die benötigten .OBJ und .LIB Dateien erzeugt.

Dieser Fehler kann ebenfalls entstehen, wenn Blinker eine temporäre Datei anlegen möchte, jedoch die Umgebungsvariable TEMP auf einen ungültigen Pfad verweist. Im Netzwerk ist es weiterhin möglich, daß jemand die Datei gelöscht hat oder Sie in diesem Verzeichnis nicht genügend Rechte besitzen.

1110: <Datei>: ungültige Objekt- oder Bibliotheksdatei

Diese Nachricht zeigt an, daß die angegebene Datei, welche eine .OBJ oder .LIB-Datei sein kann, einen ungültige Objekt-Satz enthält.

Sollte die Objekt-Datei gültig sein, dann ist sie vielleicht beschädigt. In diesem Fall müssen Sie entweder die Datei aus dem Quellcode neu erzeugen oder sie vom Installationsmedium neu installieren.

1111: <Dateiname(Modulname)> : ungültige Segment-Definition

Es gilt das gleiche wie unter Fehler Nummer 1110

1112: <Dateiname(Modulname)>: ungültige Gruppendefinition

Es gilt das gleiche wie unter Fehler Nummer 1110

1113: Segment <name> oder seine Gruppe überschreitet 64 KB

Diese Nachricht zeigt eine der folgenden Situationen an:

- Zwei zusammengehörende Bibliotheken wurden durch weitere Bibliotheken in der Skript-Datei getrennt, so daß der Code der einen Bibliothek nicht mehr den Code der anderen Bibliothek erreichen kann. Dies kann auch bei zusammengehörenden Objektdateien geschehen, die eventuell aus derselben Bibliothek extrahiert wurden und durch zu viele andere .OBJ-Dateien im Linkskript getrennt sind.
- Nicht-overlayfähiger Code kann diesen Fehler verursacht haben. Werden Bibliotheken von Dritt-Anbietern in das Programm aufgenommen, sehen Sie bitte nochmals in der 3RDPARTY.TXT-Datei nach, die sich im Lieferumfang des Blinker befindet. Dort finden Sie Empfehlungen zur Wahl des richtigen Overlay-Verfahrens.
- Zwei Bibliotheken können dasselbe Segment mit unterschiedlichen Klassennamen deklariert haben, was in den meisten Programmen nicht vorkommen sollte.
- In der Linkdatei können zwei oder mehr Bibliotheken in falscher Reihenfolge angegeben worden sein. (Siehe Abschnitt "Reihenfolge der Bibliotheken" in der Online-Hilfe)
- Es kann sein, daß das Standard-Datensegment des Programms (gewöhnlich in der MAP-Datei des Programms als DGROUP bezeichnet) die Größe von 64 KB überschritten hat. Versuchen Sie, Near-Daten in Far-Daten umzuwandeln, indem Sie große Dateneinheiten als FAR deklarieren oder den Compiler anweisen, dies automatisch zu tun, z.B. für Microsoft C/C++ mit dem Compilerschalter -Gt. Wenn Sie mit Borland C/C++ arbeiten, können Sie von der Compileroption Large Memory Model auf Huge wechseln.
- Bei der Verwendung von CA-Clipper stellen Sie erst einmal fest, welches Add On-Produkt den größten Teil des DGROUP belegt, indem Sie systematisch die einzelnen Bibliotheken entfernen und das Programm mit dem Parameter //INFO ausführen und die Werte von DS Avail vergleichen. Nachdem die problembehaftete Bibliothek gefunden wurde, nehmen Sie am besten Kontakt mit dem technischen Support des entsprechenden Herstellers auf.

Führt keine dieser Vorgehensweisen zum Erfolg und die erneute Erzeugung oder Installation der angegebenen Datei hilft ebenfalls nicht weiter, dann sollten Sie mit dem technischen Support von Blinker Kontakt aufnehmen und die genauen Einzelheiten Ihres Problems bereithalten.

Hinweis zu CA-Clipper:

Für CA-Clipper Applikationen ist der gewöhnliche Grund dieses Problems darin zu suchen, daß als erstes Objekt nicht ein CA-Clipper-Objekt angegeben wurde. Wird eine CA-Clipper-Applikation erstellt, muß das erste Objekt das CA-Clipper-Startprogramm sein.

Ist aber das erste .OBJ definitiv eine CA-Clipper-Datei, so sehen Sie bitte im Abschnitt "Reihenfolge von Bibliotheken" der Online-Hilfe nach, wo die Reihenfolge der Bibliotheken von Dritt-Anbietern in Link-Skriptdateien beschribt.

.

1114: <Datei(Modul)> : ‘<Symbol>’ ist dupliziert in <Datei(Modul)>

Es gilt die gleiche Information, wie zu Warnung 1004; jedoch ist in diesem Fall der Fehler, der aus diesem doppelt definierten Symbol resultiert so schwerwiegend, daß der Link-Vorgang sofort abgebrochen wird.

Um das Problem zu lösen, muß einem der Symbole ein anderer Name gegeben werden und die zugehörige .OBJ-Datei neu kompiliert werden.

1114: DEFINE Befehl: '<Symbol>' ist dupliziert in <Datei(Modul)>

Es gilt die gleiche Information, wie zu Warnung 1004; jedoch ist in diesem Fall der Fehler, der aus diesem doppelt definierten Symbol resultiert so schwerwiegend, daß der Link-Vorgang sofort abgebrochen wird. Um das Problem zu lösen, muß entweder der DEFINE Befehl entfernt werden, oder dem Symbol ein anderer Name gegeben werden und die zugehörige .OBJ-Datei neu kompiliert werden.

1115: <Datei(Modul)>: '<Symbol>' nicht aufgelöstes External

Diese Nachricht zeigt an, daß das in dem angegebenen Modul extern definierte '<Symbol>' der Datei <Datei> in keiner der angegebenen Objekt-Dateien oder Bibliotheken gefunden werden konnte.

Der übliche Ursache dieses Problems ist meist, daß eine Objekt-Datei oder Bibliothek in der Skript-Datei fehlt.

Eine andere mögliche Ursache ist, daß einige Bibliotheken die Verwendung des SEARCH-Befehls erfordern, weil ihre Cross-Referenz-Sätze unvollständig sind. Versuchen Sie ein SEARCH auf die Bibliothek, von der Sie annehmen, daß Sie das fehlende Symbol enthält. Eine vollständige Beschreibung des SEARCH-Befehls können Sie in der Online-Hilfe im Abschnitt "Blinker Befehle" ersehen.

Sollten Sie sich über die Gründe für diesen Fehler im klaren sein und dennoch mit dem Linken fortfahren wollen, so verwenden Sie bitte das BLINKER EXECUTABLE NODELETE Kommando.

Hinweis für CA-Clipper Sommer '87:

Sollten Sie das Symbol '__dos_hold' als unaufgelöstes Symbol erhalten, so überprüfen Sie bitte, ob Sie die letzte Version von CLIPPER.EXE, CLIPPER.LIB und EXTEND.LIB von CA-Clipper Sommer '87 besitzen. Die Original-Versionen tragen alle das Datum vom 21.12.87 und die Uhrzeit 12:00. Die letzten Versionen (also die, die Sie benötigen) haben das gleiche Datum, jedoch ist die Uhrzeit auf 2.00am gestellt. Blinker arbeitet nur mit dieser letzten Version korrekt, da eine ganze Reihe an Fehlern darin beseitigt wurden.

1116: Fixup-Überlauf bei Offset <Offset> in Segment in <Modul> mit Bezug auf <Symbol> in Modul <Modul>

Diese Meldung gibt die genaue Stelle im Programm an, wo eine Near- Referenz auf ein Symbol erfolgt, das weiter als 64KB entfernt ist. Dies kann in mehreren Situationen vorkommen:

- Zwei zusammengehörende Bibliotheken wurden durch weitere Bibliotheken in der Skript-Datei getrennt, so daß der Code der einen Bibliothek nicht mehr den Code der anderen Bibliothek erreichen kann. Dies kann auch bei zusammengehörenden Objektdateien geschehen, die eventuell aus derselben Bibliothek extrahiert wurden und durch zu viele andere .OBJ-Dateien im Linkskript getrennt sind.
- Nicht-overlayfähiger Code kann diesen Fehler verursacht haben. Werden Bibliotheken von Dritt-Anbietern in das Programm aufgenommen, sehen Sie bitte nochmals in der 3RDPARTY.TXT-Datei nach, die sich im Lieferumfang des Blinker befindet. Dort finden Sie Empfehlungen zur Wahl des richtigen Overlay-Verfahrens.
- Zwei Bibliotheken können dasselbe Segment mit unterschiedlichen Klassennamen deklariert haben, was in den meisten Programmen nicht vorkommen sollte.
- In der Linkdatei können zwei oder mehr Bibliotheken in falscher Reihenfolge angegeben worden sein. (Siehe Abschnitt "Reihenfolge der Bibliotheken" in der Online-Hilfe)

Sollten diese Lösungen das Problem noch nicht beheben, dann muß entweder der Code so modifiziert werden, daß das <Symbol> mittels einer ‘far’-Anweisung adressiert wird, oder Sie müssen sicherstellen, daß das angegebene Segment und Symbol gegenseitig erreichbar sind, indem Sie ihre jeweiligen Positionen innerhalb der MAP-Datei überprüfen. Um festzustellen, welche Bibliothek das Problem verursacht, können Sie systematisch jede Bibliothek entfernen und neu linken.

1117: Für die Verwendung von Debug-Informationen sind externe Overlays erforderlich

Wenn zu Overlay-Programmen Debugger-Informationen hinzugefügt werden sollen, dann müssen mit Hilfe des Befehls SECTION INTO externe Overlays erzeugt werden. Weitere Informationen finden Sie in der Online-Hilfe im Abschnitt "Debugger Unterstützung", der über Blinker mit unterschiedlichen Debuggern handelt.

1118: zu viele externe Overlays

Diese Nachricht zeigt an, daß mehr als 15 externe Overlay-Dateien in der Link-Datei angegeben wurden.

Diese Blinker-Version unterstützt maximal 15 externe Overlay-Dateien. Beachten Sie bitte, daß der einzige sinnvolle Grund für mehrere externe Overlay-Dateien nur der sein kann, daß die einzelnen Overlay-Dateien auf eine 360-KB Diskette passen sollen, damit die Applikation ausgeliefert werden kann.

Eine komplette Beschreibung der Technik externer Overlays entnehmen Sie bitte dem Abschnitt über den Befehl SECTION INTO in der Online-Hilfe.

1119: kein Hauptmodul gefunden

Diese Nachricht zeigt an, daß keine Startprozedur in den .OBJ und .LIB-Dateien gefunden wurde. Gewöhnlich bedeutet dies, daß die sprachspezifische Bibliothek, die den Start-Code enthält, nicht im Linkskript angegeben wurde.

1120: Inkrementelle Linkdatei ist zu groß

Diese Nachricht zeigt an, daß daß die inkrementelle Datei, die durch Blinker erzeugt wird, die maximal mögliche Größe überschritten hat.

Am einfachsten läßt sich die Größe der inkrementellen Linkdatei dadurch reduzieren, daß CA-Clipper .OBJ-Dateien, die sich selten ändern, mittels Bibliotheksverwaltungsprogrammen wie Microsoft LIB.EXE oder Borland TLIB.EXE in eine Bibliothek gestellt werden. Denn für Bibliotheken werden keine inkrementellen Informationen gespeichert, sodaß die inkrementelle Linkdatei kleiner wird.

Alternativ kann man auch .CLP-Dateien verwenden, um größere .OBJ-Dateien zu erstellen. Aber ein Compile- und Link-Lauf würde insgesamt langsamer werden, da längere Kompilierzeiten nötig sind.

1121: Bei BASIC muß die erste .OBJ-Datei in das Root

Für Microsoft BASIC ist es erforderlich, daß sich die .OBJ-Hauptdatei eines Programms im Root befindet, d.h. außerhalb eines BEGIN/ENDAREA-Konstrukts. Alle anderen .OBJ-Dateien von BASIC sind overlayfähig.

1122: <Datei(Modul)> : mit Sommer '87 und nicht mit 5.x kompiliert

Diese Nachricht zeigt an, daß Blinker entdeckt hat, daß das gerade bearbeitete Modul mit CA-Clipper Sommer ’87 kompiliert wurde, das Start-Modul jedoch ein CA-Clipper 5.x-Programm ist. Alle Module einer CA-Clipper Applikation müssen mit der gleichen Compilerversion von CA-Clipper erstellt worden sein.

Stellen Sie sicher, daß nur die korrekten Versionen der CA-Bibliotheken und der von Dritt-Anbietern

gelinkt werden. Stellen Sie ebenfalls sicher, daß alle .OBJ-Dateien mit der gleichen CA-Clipper-Version erstellt wurden.

1123: <Datei(Modul)>: mit 5.x und nicht mit Sommer '87 kompiliert

Diese Nachricht zeigt an, daß Blinker entdeckt hat, daß das gerade bearbeitete Modul mit CA-Clipper 5.x kompiliert wurde, daß Start-Modul jedoch ein CA-Clipper Sommer ’87 Programm ist. Alle Module einer CA-Clipper Applikation müssen mit der gleichen Compilerversion von CA-Clipper erstellt worden sein.

Stellen Sie sicher, daß nur die korrekten Versionen der CA-Clipper-Bibliotheken und Drittanbieter-Bibliotheken gelinkt werden. Stellen Sie ebenfalls sicher, daß alle .OBJ-Dateien mit der gleichen CA-Clipper Version erstellt wurden.

1124: <Datei(Modul)>: erste .OBJ-Datei nicht mit CA-Clipper kompiliert

Innerhalb eine CA-Clipper-Applikation ist das erste durch FILE ausgewiesene Objekt kein mit CA-Clipper erstelltes Modul. In einer CA-Clipper-Applikation muß die erste .OBJ-Datei ein CA-Clipper-Startprogramm sein.

Sollte dieser Fehler auftreten, so reorganisieren Sie Ihre Link-Datei so, daß der erste FILE-Befehl ein CA-Clipper-Modul enthält.

1125: <Datei(Modul)> kommunale Variable ‘<Symbol>’ hat eine inkonsistente Elementgröße

Diese Nachricht wird durch C-Module ausgelöst. Blinker hat erkannt, daß die Größe eines uninitialisierten ‘far-static’-Datenelements (kommunale Variable) vorher in einem anderen Modul mit einer anderen Größe deklariert worden war.

Sollten Sie diese Nachricht erhalten, so ändern Sie die C-Quellen so, daß die kommunale Variable in allen Modulen mit der gleichen Größe deklariert wird.

1126: <Datei(Modul)> kommunale Variable ‘<Symbol>’ hat einen ungültigen Datentyp

Diese Nachricht wird durch C-Module ausgelöst. Blinker hat erkannt, daß der Typ der kommunalen Variable weder NEAR noch FAR ist.

Sollten Sie diese Nachricht erhalten, ist wahrscheinlich die angegebene Datei defekt. Rekompilieren oder reinstallieren Sie das Modul erneut.

1127: Segment mit NEAR COMMUNALS überschreitet 64 KB

Die Gesamtgröße des Segments der NEAR kommunalen Variablen überschreitet die maximal mögliche Größe von 64KByte.

Um die Größe des Segments zu verringern, ändern Sie im Programm einige uninitialisierte NEAR Datendeklarationen zu FAR.

1128: <Datei(Modul)>: unbekannter OBJ-Recordtyp <nn>h

Blinker entdeckte einen Objekt-Satztyp, der zur Zeit nicht unterstützt wird. Es ist möglich, daß das Objekt defekt ist.

Sollte eine Reinstallation oder Rekompilierung nichts nützen, so benachrichtigen Sie bitte den technischen Support und halten Sie Ihre Seriennummer und eine Beschreibung des Problems parat.

Es ist auch möglich, daß Blinker auf einen neuen Objektdatentyp gestoßen ist; also vielleicht auf einen, der durch eine neue Compiler-Version kreiert wurde und von der genutzten Blinker-Version noch nicht

unterstützt wird. Sollten Sie nicht mit der neuesten Version von Blinker arbeiten, dann sollten Sie unbedingt ein Upgrade anfordern.

1129: BLINKER OVERLAY DYNAMIC ist nur für Clipper Sommer '87 gültig

Blinker unterstützt die dynamische Overlay-Belegungsmethode (wobei Speicher für Overlays von CA-Clippers FREE POOL belegt wird) nur für CA-Clipper Sommer ’87 ,und die gelinkte Applikation entspricht nicht dieser Version.

Entfernen Sie die Anweisung BLINKER OVERLAY DYNAMIC aus der Linkskriptdatei.

1130: kann Datei <Dateiname> nicht erzeugen

Sollte dieser Fehler auftreten, kann dies daran liegen, daß es diese Datei schon auf dem angegebenen Pfad gibt und sie mit dem Attribut ‘Read-Only’ (nur lesen) versehen ist. Es kann auch sein, daß nicht genügend Schreibrechte zu dem Pfad vorhanden sind, oder daß die Platte voll ist.

1131: kann Datei <Dateiname> nicht schließen

Sollte dieser Fehler auftreten, kann dies daran liegen, daß ein anderer Benutzer die Datei zu dem Zeitpunkt gelöscht hat, zu dem Blinker noch mit der Datei arbeitete. Ein anderer möglicher Grund kann ein Netzwerk oder Plattenfehler sein.

1132: kann Datei <Dateiname> nicht löschen

Tritt dieser Fehler auf, so wird die Datei wahrscheinlich noch andersweitig verwendet, oder es sind keine Löschrechte vorhanden. Ebenso kann es sein, daß die Datei in einem Verzeichnis liegt, das mit dem DOS-Befehl APPEND oder dem Netzwerkbefehl SEARCH definiert wurde.

1133: Version <n.n> wird zum Einsatz dieser .LNK-Datei benötigt

Diese Meldung bedeutet, das die .LNK-Datei eine Version des Blinkers verlangt, die neuer ist als die momentan benutzte.

1134: kann Datei <Dateiname> nicht lesen

Blinker war nicht in der Lage, aus der angegebenen Datei zu lesen. Überprüfen Sie bitte, ob auf die Datei über den gesetzten PATH oder andere Verzeichniseinstellungen der Umgebungsvariablen zugegriffen werden kann, oder tragen Sie eine vollständige Pfadangabe in die Linkdatei ein. Die Datei könnte auch in Benutzung sein oder beschädigt sein.

1135: kann Datei <Dateiname> nicht schreiben

Blinker war nicht in der Lage, in die angegebene Datei zu schreiben. Überprüfen Sie die Größe des freien Plattenspeichers. Untersuchen Sie weiterhin, ob nicht eine nichtlöschbare Datei gleichen Namens die Operationen abblockt.

1136: keine weiteren XMS-Handles verfügbar

Diese Nachricht zeigt an, daß für den von Blinker während des Linkens genutzten virtuellen Speichers keine weiteren XMS-Handles mehr vergeben werden konnten. Erhöhen Sie bitte die Anzahl der dem XMS-Speichermanager zur Verfügung stehenden Handles.

1137: keine weiteren EMS-Handles verfügbar

Diese Nachricht zeigt an, daß für den von Blinker während des Linkens genutzten virtuellen Speicher keine weiteren EMS-Handles mehr vergeben werden konnten. Erhöhen Sie bitte die Anzahl der dem

EMS-Speichermanager zur Verfügung stehenden Handles.

1138: kann Programm nicht finden und starten

Das für die Arbeit mit CodeView Version 4 erforderliche Programm CVPACK.EXE konnte nach dem Linken nicht aufgefunden werden. Symantec C++ enthält ein ähnlich benanntes Dienstprogramm, daß in CVPACK.EXE kopiert werden sollte.

1139: unausgeglichenes Verhältnis von DEFBEGINs und DEFENDs

Diese Nachricht gibt an, daß die Anzahl der in der Moduldefinitionsdatei(en) gefundenen DEFBEGIN- und DEFEND-Befehle nicht gleich ist. Dadurch wird die Linkdatei ungültig. Berichtigen Sie den Fehler und starten Sie einen neuen Versuch.

1140: neue .EXE-Datei enthält mehr als 254 Segmente

Diese Nachricht weist darauf hin, daß die Windows oder DOS Extended .EXE- oder .DLL-Datei mehr als die 254 Segmente enthält, die in einem Microsoft NEW .EXE Format maximal möglich sind Diesen Fehler kann man normalerweise mit dem Linkbefehl PACKDATA beheben, der beim Linken, soweit möglich, mehrere Datensegmente kombiniert.

Dieser Fehler kann auch auftreten, wenn die Größe der zusammengesetzten Segmente über den Befehl BLINKER SEGMENT THRESHOLD reduziert wurde. Der Wert für THRESHOLD muß wieder erhöht werden, bis die .EXE-Datei erfolgreich erstellt werden kann.

1141: neue .EXE-Datei braucht größeren ALIGNMENT-Wert

Diese Nachricht zeigt an, daß die Microsoft NEW .EXE-Datei (DOS Extended oder 16Bit Windows .EXE oder .DLL) größer als 65535 mal dem (über den Befehl BLINKER EXECUTABLE ALIGNMENT nnn angegebenen oder Standard) Ausrichtungswert der .EXE-Datei werden würde. Dies ist eine Besonderheit des NEW .EXE Formats, und der Standard Ausrichtungswert ist 64 für CA-Clipper Programme und 16 für alle übrigen Programme.

Diesen Fehler kann man beheben durch Hinzufügen des Befehls BLINKER EXECUTABLE ALIGNMENT nnn zum Linkskript und successive Erhöhung des Wertes in Potenzen von 2, z.B. 32, 64 etc., bis die Ausgabedatei erfolgreich erzeugt wird.

Beachten Sie, daß Blinker 5.0 diese Meldung anzeigen kann beim Linken von großen Programmen, die mit früheren Versionen fehlerfrei gelinkt wurden, aber dem Grenzwert nahe kamen. Das ist aber ganz normal und sollte wie oben beschrieben behoben werden.

In einigen besonderen Fällen läßt sich dieser Fehler auch mit dem Befehl PACKDATA beheben, der während des Linkens nach Möglichkeit mehrere Datensegmente zu einem kombiniert.

1142: <Dateiname(Modulname)> : gehört schon zu Gruppe <Gruppe>

Diese Nachricht zeigt an, daß ein Segment in dem angegebenen Modul in zwei verschiedenen Gruppen definiert wurde, was nicht gestattet ist.

1143: nicht genügend Hauptspeicher für Komprimierungssystem

Diese Meldung bedeutet, daß Blinker für das Komprimierungssystem nicht genügend Speicher reservieren konnte. Unter normalen Umständen sollte dieser Fehler nicht vorkommen.

Wenn dieser Fehler auftritt, stellen Sie dem DOS-Prompt mehr Extended Speicher (DPMI oder XMS) zur Verfügung durch Einstellung der Maximalwerte in den Eigenschaften der DOS-Box. Wenn dies nichts hilft, vergrößern Sie in der Systemsteuerung die maximale Größe der Windows Swapdatei und prüfen Sie,

ob auf der Platte auch ausreichend Platz dafür verfügbar ist. Schließlich können Sie noch versuchen, das Programm mit der Windows-Version des Blinkers neu zu linken.

1144: die Summe von DGROUP- und Heap-Größe überschreitet 64 KB

Diese Nachricht gibt an, daß die Größe des Heap 64 KB überschreitet, wenn sie zu der Größe des Datensegments DGROUP einschließlich des Stack addiert wird.

Die Größe des Heap kann mit Hilfe des Moduldefinitionsbefehls HEAPSIZE nnnn gesetzt oder reduziert werden. Über HEAPSIZE MAXVAL können Sie sie auf die maximal mögliche Größe setzen.

1145: Symbol <Symbol> verwendet eine doppelt vergebene Ordnungsnummer <Ordnungsnummer>

Diese Nachricht gibt an, daß angegebene Symbol exportiert werden sollte, die für den Export spezifizierte Ordnungsnummer jedoch bereits von einen anderen exportierten Symbol verwendet wird.

1146: Der Befehl APPLOAD muß vor den SEGMENTS-Befehlen genannt werden

Diese Nachricht zeigt, an, daß in der Moduldefinitionsdatei nach einem oder mehreren SEGMENTS-Befehlen ein APPLOAD-Befehl gefunden wurde. Das ist nicht gestattet.

1147: <Dateiname(Modulname)> enthält inkrementelle Kompilierungsfehler

Diese Nachricht deutet darauf hin, daß die angegebene .OBJ-Datei mit einem Compiler erzeugt wurde, der die inkrementelle Kompilierung unterstützt, die Kompilierung jedoch fehlerhaft ausgeführt wurde. Korrigieren Sie den Fehler und kompilieren Sie die angegebene .OBJ-Datei erneut. Danach können Sie einen weiteren Linkversuch starten.

1148: BLINKER CLIPPER SYMBOL OFF bei sehr großer Symboltabelle verwenden

Diese Meldung kommt nur bei Windows-Programmen vor, die mit CA-Clipper 5.x geschrieben wurden. Sie besagt, daß die Symboltabelle zu groß ist, um komprimiert zu werden. Dies ist auf Beschränkungen in Laufzeitmodulen der CLIPPER.LIB zurückzuführen.

Fügen Sie einfach den Befehl BLINKER CLIPPER SYMBOL OFF am Ende Ihrer Linkskriptdatei hinzu, um die Fehlermeldung wegzubekommen und die .EXE-Datei zu erzeugen.

1149: komprimierte Programme können nicht inkrementell gelinkt werden

Diese Meldung sagt, daß inkrementelles Linken explizit mit der Anweisung BLLINKER INCREMENTAL ON im Linkskript eingeschaltet wurde. Dies aber wird für komprimierte Programme nicht unterstützt.

Entfernen Sie die explizite BLINKER INCREMENTAL ON Anweisung aus dem Linkskript.

1150: <Dateiname> ist eine Demo und kann daher nicht mit diesem Blinker gelinkt werden

Diese Meldung besagt, daß die angegebene Bibliothek eine spezielle Demonstrations-Bibliothek ist, die nur mit einer Demo-Version von Blinker gelinkt werden kann. Die Demo-Version von Blinker sollte mit der Demonstrations-Bibliothek ausgeliefert worden sein.

1151: Parameter PACKDATA erhöhen, um SYMBOL-Segmente zu reduzieren

Diese Meldung besagt, daß in einem inkrementell gelinkten CA-Clipper-Programm zu viele SYMBOL-Segmente waren , sodaß es nötig ist, den Parameter im PACKDATA-Befehl in der Link-Skriptdatei zu

erhöhen oder hinzuzufügen.

1152: zu viele exportierte Namen-Symbole

Diese Meldung besagt, daß die .EXE- oder .DLL-Datei, die Sie erzeugen, zu viele exportierte Namen-Symbole enthält, um in den Header der .EXE-Datei eingetragen werden zu können.

Entweder entfernen Sie einige der EXPORT-Befehle aus der Moduldefinitionsdatei oder entfernen Sie eine oder mehrere Import-Bibliotheken aus der Link-Skriptdatei.

1153: n.n erfordert eine neuere Version von BLXxxxnn.LIB

Diese Meldung weist darauf hin, daß die angegebene Bibliothek von einer älteren Version des Blinker stammt.

Überpüfen Sie, ob die Linkskriptdatei und der Umgebungspfad LIB auf die neueste Version der angegebenen Bibliothek zeigen.

1154: portierbare .EXE-Datei enthält mehr als 128 Abschnitte

Diese Meldung zeigt an, daß die portierbare .EXE- oder .DLL-Datei mehr als 128 Abschnitte enthält.

Sehen Sie in Ihrer Compiler-Dokumentation nach, wie die Anzahl der Abschnitte in der .EXE-Datei reduziert werden kann. Eine Möglichkeit Abschnittsnamen zu identifizieren ist, .OBJ-Dateien nach und nach in der Linkskript-Datei auszukommentieren, bis eine .EXE-Datei erzeugt werden kann, und danach anhand der MAP-Datei die Abschnittsnamen zu finden.

1155: portierbare .EXE-Datei benötigt größeren Wert für Dateiausrichtung

Diese Meldung zeigt an, daß die portierbare .EXE-Datei zu groß für die aktuelle Dateiausrichtungsgröße ist.

Mit der Anweisung BLINKER EXECUTABLE ALIGNMENT kann die Dateiausrichtungsgröße erhöht werden.

1156: <Dateiname>(<Modulname>) ist nicht kompatibel zu 32-Bit .EXE-Dateien

Diese Meldung zeigt an, daß das angegebene Modul in der angegebenen Datei Code oder Daten enthält, der nicht in einer 32-Bit-Anwendung verwendet werden kann. Der Code muß für die Verwendung von 32-Bit-Adressen kompiliert werden, bevor er mit einem 32-Bit-Programm eingesetzt werden kann.

Sind Sie im Besitz des Quelltextes für dieses Modul, dann kompilieren Sie es erneut mit der 32-Bit-Option für Ihren Compiler. Wenn nicht, fragen Sie den Hersteller des Moduls nach einer 32-Bit-Version.

1157: Importieren aus DLLs im Dual Mode nicht unterstützt

Diese Meldung zeigt an, daß versucht wurde, ein Symbol aus einer DOS-Extender-DLL in ein Dual Mode-Programm zu importieren. Dies wird nicht unterstützt.

Linken Sie das Programm neu als Extended Mode Programm, um DLLs verwenden zu können.

1158: <Dateiname> ist keine gültige Ressource-Datei

Diese Meldung zeigt an, daß die angegebene Datei keine gültige Ressource-Datei ist. Es könnte bedeuten, daß die Datei defekt ist.

Erzeugen Sie über einen Ressource-Compiler aus der .RC-Datei die Ressource-Datei neu.

1159: <Dateiname> ist eine 32-Bit-Ressource-Datei

Diese Meldung zeigt an, daß es sich bei der angegebenen Datei um eine 32-Bit-Ressource-Datei handelt, die nicht in ein 16-Bit-Windowsprogramm gelinkt werden kann.

Verwenden Sie einen 16-Bit-Ressource-Compiler, um eine 16-Bit-Ressource-Datei aus der .RC-Datei zu erzeugen.

1160: <Dateiname> ist eine 16-Bit-Ressource-Datei

Diese Meldung zeigt an, daß es sich bei der angegebenen Datei um eine 16-Bit-Ressource-Datei handelt, die nicht in ein 32-Bit-Windowsprogramm gelinkt werden kann.

Verwenden Sie einen 32-Bit-Ressource-Compiler, um eine 32-Bit-Ressource-Datei aus der .RC-Datei zu erzeugen.

1161: mehr als ein Moduldefinitionsbefehl NAME oder LIBRARY

Diese Meldung zeigt an, daß mehrere NAME oder LIBRARY Moduldefinitionsbefehle im Linkskript oder in der Moduldefinitionsdatei angegeben sind. Je Modul ist nur eine NAME oder LIBRARY-Anweisung erlaubt.

Entfernen Sie den doppelten oder extra vorhandenen Moduldefinitionsbefehl.

1162: <Dateiname> darf nicht mit einem Windows-Programm gelinkt werden

Diese Meldung zeigt an, daß die angegebene Datei eine Blinkerdatei speziell für den DOS-Extender ist und daher nicht mit einem Windows-Programm gelinkt werden kann.

Entfernen Sie die Anweisung FILE oder LIBRARY, die auf die angegebene Datei verweist.

1163: .DLLs können nicht als Dual Mode-Programme erzeugt werden

Diese Meldung zeigt an, daß versucht wurde, eine DLL im Dual Mode für den DOS-Extender zu erzeugen. Dies wird nicht unterstützt.

Linken Sie die DLL als Windows-Programm oder als Programm für den DOS-Extender.

1164: 32 Bit Windows Programm kann nicht komprimiert werden

Diese Meldung besagt, daß versucht wurde, ein 32 Bit Windows Programm zu komprimieren, was zur Zeit nicht unterstützt wird.

1165: <Dateiname> ist eine Datei im COFF Format und wird zur Zeit nicht unterstützt.

Diese Meldung besagt, daß die angegebene Datei im COFF Format statt im Intel OMF Format vorliegt und zur Zeit nicht unterstützt wird.

Blinker arbeitet mit jedem Compiler, der .OBJ und .LIB Dateien im Intel OMF Format erzeugt, was fast alle Compiler einschließt außer Microsoft Visual C++ 2.x und später, der sie im COFF Format erzeugt und deshalb zur Zeit nicht unterstützt wird.

Wenn Sie Alaska Xbase++ verwenden, achten Sie darauf, daß in der PBUILD Projektdatei OBJ_FORMAT auf OMF eingestellt ist und alle .OBJ und .LIB Dateien neu erzeugt worden sind. Stellen Sie auch sicher, daß alle separat kompilierten .OBJ Dateien mit dem Parameter /OMF kompiliert werden, und daß die LIB Umgebungsvariable auf das OMF Unterverzeichnis vor dem LIB Verzeichnis zeigt, um sicher zu gehen, daß die OMF Version der Xbase++ Bibliotheken verwendet wird. Falls im Linkskript eines PBUILD Projekts der Pfad einer Bibliothek explizit angegeben wird, muß er auch auf die OMF Version der Bibliothek zeigen.

1169: DPMI Speicherzuweisungsfehler

Diese Meldung zeigt an, daß bei der Speicherzuweisung vom DPMI-Host ein Fehler aufgetreten ist, was unter normalen Umständen nicht passieren darf. Kontaktieren Sie den technischen Support von Blink, Inc mit Ihrer Blinker-Seriennummer und exakten Angaben zu den Umständen, wie es zu diesem Fehler kam.

1170: Fehler im Komprimierungssystem

Diese Meldung zeigt an, daß bei der Komprimierung der Ausgabedatei ein Fehler aufgetreten ist, was unter normalen Umständen nicht passieren sollte. Kontaktieren Sie den technischen Support von Blink, Inc mit Ihrer Blinker-Seriennummer und exakten Angaben zu den Umständen, wie es zu diesem Fehler kam.

1171: Grenze für .EXE-Dateigröße überschritten

Diese Nachricht gibt an, daß die Größe der .EXE-Datei die interne Größenbeschränkung des Blinker überschreitet. Nehmen Sie bitte Kontakt zum technischen Support von Blink, Inc auf und geben Sie Ihre Blinker-Seriennummer sowie die genauen Umstände an, durch die der Fehler hervorgerufen wurde.

1172: Ein nicht näher spezifizierter XMS-Fehler ist aufgetreten

Das Auftreten des Fehlers 1172 kann für den Zeitraum, bis die Quelle des XMS-Problems gefunden wurde, temporär verhindert werden, indem die Verwendung von XMS durch den Blinker unterbunden wird. Dies geschieht über den Befehl BLINKER LINK XMS OFF. Dadurch kann das Problem zwar besser identifiziert, aber nicht wirklich gelöst werden.

1173: Ein nicht näher spezifizierter EMS-Fehler ist aufgetreten

Das Auftreten des Fehlers 1173 kann für den Zeitraum, bis die Quelle des EMS-Problems gefunden wurde, temporär verhindert werden, indem die Verwendung von EMS durch den Blinker unterbunden wird, Dies geschieht über den Befehl BLINKER LINK EMS OFF. Dadurch kann das Problem zwar besser identifiziert, aber nicht wirklich gelöst werden.

1174: ungültige Dateianzahl

Dieser Fehler sollte normalerweise nicht vorkommen. Wenn Sie ihn erhalten, nehmen Sie bitte Kontakt zum technischen Support von Blink, Inc auf und geben Sie Ihre Blinker-Seriennummer sowie die genauen Umstände an, durch die der Fehler hervorgerufen wurde.

1175: zu viele gepufferte Dateien

Dieser Fehler sollte normalerweise nicht auftreten. Wenn Sie diese Meldung erhalten, wenden Sie sich bitte an den technischen Support und halten Sie bitte Ihre Seriennummer und Details zu den Fehlerumständen bereit.

1176: ungültiges Symbol-Token

Blinker ist auf eine ungültige CA-Clipper Symbol-Abkürzung gestoßen, während er versuchte, die Symboltabelle zu komprimieren. Die häufigste Ursache für diesen Fehler ist eine defekte .OBJ oder .LIB-Datei.

Rekompilieren Sie das Programm und linken Sie die Applikation mit der Option VERBOSE, um das fehlerbehaftete Modul einzukreisen.

1177: ungültige Nummer von gepacktem Symbol

Blinker ist auf ein Problem während der Komprimierung der Symbol-Tabelle einer CA-Clipper-Applikation gestoßen. Der gewöhnliche Grund dieses Fehlers liegt in einer defekten .OBJ oder .LIB-

Datei.

Rekompilieren Sie das Programm und linken Sie die Applikation mit der Option VERBOSE, um das fehlerbehaftete Modul einzukreisen.

1178 ungültige Nummer von LOCAL-Symbol

Blinker ist auf ein Problem während der Komprimierung der Symbol-Tabelle eine CA-Clipper-Applikation gestoßen. Der gewöhnliche Grund dieses Fehlers liegt in einer defekten .OBJ oder .LIB-Datei.

Rekompilieren Sie das Programm und linken Sie die Applikation mit der Option VERBOSE, um das fehlerbehaftete Modul einzukreisen.

Dieser Fehler kann auch dann auftreten, wenn die Bibliotheken zu CA-Clipper 5.x umbenannt wurden und stattdessen automatisch die CA-Clipper Sommer ’87 Bibliotheken eingebunden werden. In diesem Fall verwenden Sie bitte den NODEFLIB-Befehl.

1179: zu viele Einträge in gepackter Symboltabelle

Blinker ist beim Packen der Symboltabelle eines CA-Clipper- Programms an eine interne Grenze gestoßen. Dies ist unter normalen Umständen sehr unwahrscheinlich.

Wenden Sie sich bitte an den technischen Support und halten Sie Ihre Seriennummer sowie Details zu den Umständen des Fehlers bereit.

1180: ungültige Nummer von Communal

Die Fehler 1180-1187 zeigen an, daß Blinker eine interne Größenbegrenzung für die entsprechende Tabelle überschritten hat.
Haben Sie eine dieser Nachrichten erhalten, sollten Sie sich erst einmal die neueste Version von Blinker beschaffen, damit Sie mit der aktuellsten größten internen Tabelle arbeiten können. Wenn eine Aktualisierung der Version Ihr Problem nicht aus der Welt geschafft hat, dann sollten Sie mit dem technischen Support von Blinker Kontakt aufnehmen und diesem zusammen mit der Seriennummer Ihres Blinker auch die genauen Einzelheiten der Umstände vorlegen, durch die der Fehler hervorgerufen wurde.

1181: zu viele Einträge in LNAME-Tabelle

Die Fehler 1180-1187 zeigen an, daß Blinker eine interne Größenbegrenzung für die entsprechende Tabelle überschritten hat.
Haben Sie eine dieser Nachrichten erhalten, sollten Sie sich erst einmal die neueste Version von Blinker beschaffen, damit Sie mit der aktuellsten größten internen Tabelle arbeiten können. Wenn eine Aktualisierung der Version Ihr Problem nicht aus der Welt geschafft hat, dann sollten Sie mit dem technischen Support von Blinker Kontakt aufnehmen und diesem zusammen mit der Seriennummer Ihres Blinker auch die genauen Einzelheiten der Umstände vorlegen, durch die der Fehler hervorgerufen wurde.

1182: zu viele Einträge in Sortiertabelle

Die Fehler 1180-1187 zeigen an, daß Blinker eine interne Größenbegrenzung für die entsprechende Tabelle überschritten hat.
Haben Sie eine dieser Nachrichten erhalten, sollten Sie sich erst einmal die neueste Version von Blinker beschaffen, damit Sie mit der aktuellsten größten internen Tabelle arbeiten können. Wenn eine Aktualisierung der Version Ihr Problem nicht aus der Welt geschafft hat, dann sollten Sie mit dem technischen Support von Blinker Kontakt aufnehmen und diesem zusammen mit der Seriennummer Ihres Blinker auch die genauen Einzelheiten der Umstände vorlegen, durch die der Fehler hervorgerufen wurde.

1183: zu viele Einträge in Symbolnamen-Tabelle

Die Fehler 1180-1187 zeigen an, daß Blinker eine interne Größenbegrenzung für die entsprechende Tabelle überschritten hat.
Haben Sie eine dieser Nachrichten erhalten, sollten Sie sich erst einmal die neueste Version von Blinker beschaffen, damit Sie mit der aktuellsten größten internen Tabelle arbeiten können. Wenn eine Aktualisierung der Version Ihr Problem nicht aus der Welt geschafft hat, dann sollten Sie mit dem technischen Support von Blinker Kontakt aufnehmen und diesem zusammen mit der Seriennummer Ihres Blinker auch die genauen Einzelheiten der Umstände vorlegen, durch die der Fehler hervorgerufen wurde.

1184: zu viele Einträge in der dynamischen Tabelle

Die Fehler 1180-1187 zeigen an, daß Blinker eine interne Größenbegrenzung für die entsprechende Tabelle überschritten hat.
Haben Sie eine dieser Nachrichten erhalten, sollten Sie sich erst einmal die neueste Version von Blinker beschaffen, damit Sie mit der aktuellsten größten internen Tabelle arbeiten können. Wenn eine Aktualisierung der Version Ihr Problem nicht aus der Welt geschafft hat, dann sollten Sie mit dem technischen Support von Blinker Kontakt aufnehmen und diesem zusammen mit der Seriennummer Ihres Blinker auch die genauen Einzelheiten der Umstände vorlegen, durch die der Fehler hervorgerufen wurde.

1185: Platz für dynamische Tabelle ist voll

Die Fehler 1180-1187 zeigen an, daß Blinker eine interne Größenbegrenzung für die entsprechende Tabelle überschritten hat.
Haben Sie eine dieser Nachrichten erhalten, sollten Sie sich erst einmal die neueste Version von Blinker beschaffen, damit Sie mit der aktuellsten größten internen Tabelle arbeiten können. Wenn eine Aktualisierung der Version Ihr Problem nicht aus der Welt geschafft hat, dann sollten Sie mit dem technischen Support von Blinker Kontakt aufnehmen und diesem zusammen mit der Seriennummer Ihres Blinker auch die genauen Einzelheiten der Umstände vorlegen, durch die der Fehler hervorgerufen wurde.

1186: zu viele FIXUPs im Overlay

Die Fehler 1180-1187 zeigen an, daß Blinker eine interne Größenbegrenzung für die entsprechende Tabelle überschritten hat.
Haben Sie eine dieser Nachrichten erhalten, sollten Sie sich erst einmal die neueste Version von Blinker beschaffen, damit Sie mit der aktuellsten größten internen Tabelle arbeiten können. Wenn eine Aktualisierung der Version Ihr Problem nicht aus der Welt geschafft hat, dann sollten Sie mit dem technischen Support von Blinker Kontakt aufnehmen und diesem zusammen mit der Seriennummer Ihres Blinker auch die genauen Einzelheiten der Umstände vorlegen, durch die der Fehler hervorgerufen wurde.

1187: nicht genügend virtueller Speicher zum Linken vorhanden

Die Fehler 1180-1187 zeigen an, daß Blinker eine interne Größenbegrenzung für die entsprechende Tabelle überschritten hat.
Haben Sie eine dieser Nachrichten erhalten, sollten Sie sich erst einmal die neueste Version von Blinker beschaffen, damit Sie mit der aktuellsten größten internen Tabelle arbeiten können. Wenn eine Aktualisierung der Version Ihr Problem nicht aus der Welt geschafft hat, dann sollten Sie mit dem technischen Support von Blinker Kontakt aufnehmen und diesem zusammen mit der Seriennummer Ihres Blinker auch die genauen Einzelheiten der Umstände vorlegen, durch die der Fehler hervorgerufen wurde.

1188: .EXE Datei ist defekt

Diese Nachricht besagt, daß die Datei BLINKER.EXE defekt ist. Bitte installieren Sie Blinker neu.

1189: ungültiger Seitenstatus

1190: Interner Fehler - ungültige Symbolnummer

1191: Interner Fehler - ungültige Bibliotheksnummer

1192: Interner Fehler - ungültige Seitennummer

1193: Interner Fehler - ungültige CodeView-Nummer

1194: Interner Fehler - ungültige Modulnummer

1195: Interner Fehler - ungültige PUBLIC-Nummer

1196: Interner Fehler - ungültige External-Nummer

1197: Interner Fehler - ungültige Gruppennummer

1198: Interner Fehler - ungültige Segment-Nummer

1199: Interner Fehler - Fehler bei DOS-Funktionsaufruf

Die Fehlermeldungen 1189-1199 dürften bei normalen Operationen nicht aufteten.

Einer der häufigsten Gründe für diese Art Fehler ist ein Speicherkonflikt. Starten Sie das mit dem Blinker mitgelieferte Dienstprogramm BLINKMEM und suchen Sie nach solchen Problemquellen.

Ein anderer häufiger Grund für solche Fehler ist eine beschädigte Objektdatei oder Bibliothek. Linken Sie erneut und geben Sie dabei das Schlüsselwort VERBOSE an: das/die letzte aufgeführte Modul/Bibliothek (oder in einigen Fällen auch das/die nach dem/r zuletzt aufgeführten Modul/Bibliothek folgende Modul/Bibliothek (in der Reihenfolge der Linkdatei) ist höchstwahrscheinlich die betreffende Datei. Erstellen oder installieren Sie diese Datei noch einmal.

Wenn durch keine dieser Prozeduren das Problem gelöst wird, dann sollten Sie Kontakt mit dem technischen Support von Blinker aufnehmen und Ihre Seriennummer sowie die genauen Einzelheiten der Umstände angeben, durch die der Fehler hervorgerufen wird.

DOS-Real Mode-Laufzeitfehler (12xx)

Fehlermeldungen 1201 bis 1299 sind Real Mode Laufzeitmeldungen.

1201: Overlaydatei <Dateiname> nicht im aktuellen Pfad gefunden

Diese Meldung besagt, daß entweder zu viele Dateien geöffnet sind oder daß die externe Overlay-Datei <Dateiname>, die zur Linkzeit angegeben und erzeugt worden ist, nicht im aktuellen Pfad oder Verzeichnis gefunden werden kann.

Stellen Sie sicher, daß sich die Overlay-Datei entweder im aktuellen Verzeichnis, im Verzeichnis mit der .EXE Datei oder in einem Verzeichnis aus der DOS PATH Umgebungsvariablen befindet. Wenn ja, dann versuchen Sie, der .EXE- und der Overlay-Datei das Attribut NUR LESEN (read only) zu geben.

Hinweis: Die Blinker Fehler 1201 bis 1204 können manchmal auch von einem Virus erzeugt werden. Wenn die obigen Maßnahmen nicht helfen, führen Sie eine gründliche Virensuche durch und entfernen sie die gefundenen gegebenenfalls.

1202: DOS-Lesefehler in Overlaydatei <Dateiname>

Diese Meldung zeigt an, daß DOS beim Lesen der Overlay-Datei einen Fehler zurückgegeben hat.

Dieser Fehler wird häufig durch verschiedene Situationen herbeigeführt:

Dieser Fehler tritt normalerweise dann auf, wenn eine Funktion aufgerufen wird, welche die maximale Anzahl der DOS-Datei- Handles für DOS Versionen 3.0 bis 3.2 erhöht.Wenn diese 'File-Handle-Routine' aufgerufen wird, nachdem Blinker die Overlay-Datei bereits geöffnet und Overlays geladen hat, wird die File-Handle Tabelle neu erzeugt und Blinkers bestehende File-Handles für die Overlays werden gelöscht. Um dies zu vermeiden, achten Sie darauf, daß die Funktion zur Änderung der Handles aus dem Root aufgerufen wird, bevor Blinker das erste Overlay geladen hat.

Hinweis für CA-Clipper-Programmierer:

Eion Kombination von drei Faktoren kann dieses Problem in einer CA-Clipper-Anwendung bewirken:

- (a) Aufruf einer Funktionen zur Erhöhung der Anzahl der Handles (z.B. SETHANDLES(n) von FUNCky oder NHANDLES() von NetLib) 0aus einem Overlay heraus
- (b) Es wird DOS 3.3 oder eine höhere Version verwendet
- (c) Es wird der Clipper-Umgebungsparameter Fn belegt

Da die Routine zur Heraufsetzung der Handles nur für DOS-Versionen 3.0 bis 3.2 gebraucht wird, kann dieser Fehler vermieden werden, wenn man die DOS-Version überprüft, bevor man die Funktion aufruft:

```
IF DOSVERSION () < 3.3
   SETHANDLES (29)  # FUNCky-, NetLib-, Shareware-Funktion
ENDIF
```

(2) Dieser Fehler kann auch dann auftreten, wenn nicht-overlayfähiger Code in einem Overlay-Bereich plaziert worden ist. Falls Sie Objektdateien oder Bibliotheken verwenden, von denen Sie nicht wissen, ob sie overlayfähig sind, sollten Sie diese immer in den Root-Bereich legen und danach die Applikation neu linken.

Wenn die Applikation danach einwandfrei läuft, bedeutet dies, daß die betreffenden Dateien (noch) nicht overlayfähig sind. Die Datei 3RDPARTY.TXT enthält eine Übersicht mit Bibliotheken von Drittanbietern und beschreibt deren Kompatibilität und Overlay-Fähigkeit mit Blinker.

(3) Sollte der Programm-Stack überlaufen, kann er den benachbarten Datei-Handle-Bereich überschreiben und diesen Fehler hervorrufen. Versuchen Sie die Stackgröße dem Befehl STACK oder BLINKER PROCEDURE DEPTH zu verändern.

(4) Ein Netzwerk könnte so konfiguriert sein, daß es den Zugriff auf die .EXE-Datei nach dem Laden verweigert und dadurch den Overlay-Manager des Blinker hindert, ein internes Overlay zu lesen. Auch Swap-Funktionen können diesen Fehler hervorrufen, falls die Datei-Handle-Tabelle nicht korrekt wiederhergestellt wurde.

(5) Falls dieser Fehler nur sporadisch auftritt, ist eine Beurteilung der Situation wesentlich schwieriger. Möglicherweise wurde die Overlay-Datei durch externen Einfluß geschlossen, wie z.B. durch ein TSR-Programm oder durch Ausloggen aus dem Netzwerk, oder die Overlay-Datei ist wegen eine Plattenfehlers defekt und muß neu installiert werden.

Hinweis: Die Blinker Fehler 1201 bis 1204 können manchmal auch von einem Virus erzeugt werden. Wenn die obigen Maßnahmen nicht helfen, führen Sie eine gründliche Virensuche durch und entfernen sie die gefundenen gegebenenfalls.

1203: Datei <Dateiname> ist keine gültige .EXE-Datei

Diese Meldung zeigt an, daß der Header der betroffenen Datei kein gültiger .EXE-Header ist. Aus diesem Grund kann das interne Blinker-Overlay nicht innerhalb der Datei gefunden werden. Ursachen für diesen Fehler können eine Beschädigung der Datei, eine Überschreibung der ursprünglichen Datei mit einer anderen Datei gleichen Namens oder eine geänderte PATH-Variable sein, die auf eine andere Datei gleichen Namens in einem anderen Verzeichnis zeigt.

Versuchen Sie, die .EXE Datei neu zu erzeugen mit externen Overlays (mit dem SECTION INTO xxx Befehl), und der .EXE Datei und den .OVL Dateien ein ‘nur lesen’ Attribut zu geben.

Hinweis: Die Blinker Fehler 1201 bis 1204 können manchmal auch von einem Virus erzeugt werden. Wenn die obigen Maßnahmen nicht helfen, führen Sie eine gründliche Virensuche durch und entfernen sie die gefundenen gegebenenfalls.

1204: Overlaydatei <Dateiname> paßt nicht zu .EXE Datei

Diese Meldung besagt, daß die vom Overlay-Manager geöffnete Overlay-Datei entweder keine Blinker-Overlay-Datei ist, oder aber zu einem anderen Zeitpunkt als die gerade laufende .EXE-Datei erzeugt worden ist.

Dieser Fehler tritt dann auf, wenn die korrekte Version der Overlay-Datei von einer älterne Version überschrieben worden ist, die neue Version nicht im aktuellen Pfad gefunden werden kann oder der Link-Prozeß vor Erstellung einer korrekten Overlay-Datei abgebrochen wurde.

Versuchen Sie, die .EXE Datei neu zu erzeugen und der .EXE Datei und den .OVL Dateien ein ‘nur lesen’ Attribut zu geben.

Hinweis: Die Blinker Fehler 1201 bis 1204 können manchmal auch von einem Virus erzeugt werden. Wenn die obigen Maßnahmen nicht helfen, führen Sie eine gründliche Virensuche durch und entfernen sie die gefundenen gegebenenfalls.

1205: zuwenig Speicher zum Laden der Prozedur

Diese Meldung sollte nur dann auftreten, wenn CA-Clipper Sommer '87 mit dynamischer Overlayverwaltung verwendet wird (was dem Standard entspricht). Sie zeigt an, daß nicht mehr genügend dynamischer Speicher im Free Pool von CA-Clipper zur Verfügung steht, um die angeforderte Overlay-Prozedur zu laden.

Eine Beschreibung der Maximierung des für CA-Clipper verfügbaren dynamischen Speichers finden Sie in der Online Hilfe im Abschnitt "Benutzung von Blinker mit CA-Clipper".

1206: maximale Prozedurverschachtelungstiefe überschritten

Diese Meldung sagt aus, daß die Tiefe von verschachtelten Prozeduraufrufen (d.h. wiederholte Prozeduraufrufe innerhalb von Prozeduren ohne Rückkehr zur Aufrufebene) während der Ausführung des Programms das Maximum überschritten hat, das der Prozeduraufrufstack aufnehmen kann.

Die Anzahl von aufrufbaren Prozeduren können Sie ganz einfach durch Aufnahme des Befehls BLINKER PROCEDURE DEPTH <nTiefe> oder STACK <nGröße> vergrößern.

Hinweis: Die Fehler 1207 bis 1209 sind NORMALE Abschlußmeldungen und zeigen lediglich an, daß eine Demonstrationsbeschränkung überschritten wurde.

1207: nicht mehr verwendet

Diese Meldung wird nicht mehr verwendet.

1208: Datumsgrenze für Demo überschritten

Diese Meldung besagt, daß die Applikation mit dem Befehl BLINKER DEMONSTRATION DATE <Grenzdatum> gelinkt worden ist, um eine Demo-Version mit einer Beschränkung der Ausführung bis zu einem bestimmten Datum zu erstellen. Dieses Grenzdatum ist nun überschritten worden. Ausführliche Informationen zu Demo-Versionen finden Sie in der Online-Hilfe.

1209: Demo-Laufzeitlimit überschritten

Diese Meldung besagt, daß die Applikation mit dem Befehl BLINKER DEMONSTRATION MINUTES <nGrenze> gelinkt worden ist, um eine Demo-Version mit einer Beschränkung der Ausführungszeit zu erstellen. Dieses Zeitlimit ist nun überschritten worden. Ausführliche Informationen zu Demo-Versionen finden Sie in der Online-Hilfe.

1210: Overlay wurde vorzeitig aus dem Speicher entfernt

Diese Meldung kommt nur bei Clipper Sommer ’87 vor und nur wenn die dynamische Zuweisungsmethode für Overlays verwendet wird. Diese Meldung zeigt an, daß der einem Overlay zugeordnete Speicher durch die Speicherverwaltungs-Routinen von CA-Clipper vorzeitig freigegeben worden ist, was in sehr engen Speichersituationen vorkommen kann.

Dies ist ein Fehler im Clipper Sommer ’87, den der Blinker-Overlay-Manager vor jeder Manipulation von dynamischen Speicher überprüft.

Verwechseln Sie diese Meldung nicht mit dem internen Fehler 1210 von CA-Clipper 5.x.

1211: Overlaymanager-Stack-Überlauf

Der Blinker-Overlay-Manager pflegt einen ‘reload’-Stack, der zur Verwaltung der dynamischen Overlays dient. Dieser Stack ist übergelaufen. Die häufigste Ursache für diesen Fehler ist eine unkontrollierte oder ungewollte Rekursion innerhalb der Overlay-Prozedur oder Funktion. Das heißt, daß sich ein und die selbe Funktion immer wieder selbst aufruft.

Bitte erhöhen Sie die BLINKER PROCEDURE DEPTH-Einstellung, sofern diese niedrig ist, und überprüfen Sie Ihren Code nach ungewollten Rekursionen.

1212 Overlay-Betriebsgröße überschritten - Betriebsgröße erhöhen

Diese Meldung kann nur in Zusammenhang mit der BLINKER OVERLAY FIXED-Methode auftreten

und sollte nicht unter normalen Umständen erscheinen.

Normalerweise kalkuliert Blinker die minimale, benötigte OPSIZE zur Linkzeit und erlaubt nicht, daß dieser Wert auf einen Wert unter dem ermittelten gesetzt wird.

Taucht diese Meldung auf, so kann Blinker keinen Platz im Overlay-Bereich finden, in dem die Overlay-Prozedur ausgeführt werden könnte. Erhöhen Sie die OPSIZE in diesem Falle um 1-2 KByte und linken Sie die Applikation neu. Sollte der Fehler aber weiterhin erscheinen, so kontaktieren Sie bitte den technischen Support und halten Sie die Seriennummer sowie Details der Fehlerumstände bereit.

1213: Versuch eine DEFINED-Routine aufzurufen

Das DEFINE-Kommando wurde benutzt, um eine Routine temporär ‘auszublenden’, aber innerhalb der Applikation wurde genau diese Routine aufgerufen. Entfernen Sie entweder die Aufrufe dieser Routinen in Ihrem Programm, oder entfernen Sie die entsprechende DEFINE-Anweisung aus dem Linkskript.

1214: Fehler beim Zugriff auf EMS-Overlaycache

Diese Nachricht zeigt an, daß ein Fehler beim Zugriff des Blinker-Overlay-Managers auf den EMS-Puffer auftrat, welcher zur Speicherung oder Wiederverwendung von Overlays verwandt wird. Bleibt das Problem bestehen, entfernen Sie bitte die BLINKER CACHE EMS-Anweisung aus der Linkskript-Datei.

Sollte dies das Problem lösen, kann es sein, daß im Hintergrund ein TSR oder ein Gerätetreiber (Device-Driver) aktiv ist, der EMS in einer nicht üblichen Weise verwendet. Entfernen Sie alle TSR’s und Geräte-Treiber außer dem EMS-Treiber und prüfen Sie, ob der Fehler immer noch auftaucht. Sollte das Problem immer noch auftauchen, kontaktieren Sie bitte den technischen Support und halten die Seriennummer sowie Details zu den Fehlerumständen bereit.

1215: Fehler beim Zugriff auf XMS-Overlaycache

Diese Nachricht zeigt an, daß ein Fehler beim Zugriff des Blinker-Overlay-Managers auf den XMS-Puffer auftrat, welcher zur Speicherung oder Wiederverwendung von Overlays verwandt wird. Bleibt das Problem bestehen, entfernen Sie bitte die BLINKER CACHE XMS-Anweisung aus der Linkskript-Datei.

Sollte dies das Problem lösen, kann es sein, daß im Hintergrund ein TSR oder ein Geräte-Treiber (device-driver) aktiv ist, der XMS in einer nicht üblichen Weise verwendet. Entfernen Sie alle TSR’s und Geräte-Treiber außer dem XMS-Treiber und prüfen Sie, ob der Fehler immer noch auftaucht. Sollte das Problem immer noch auftauchen, kontaktieren Sie bitte den technischen Support und halten Sie die Seriennummer sowie Details zu den Fehlerumständen bereit.

1216: Overlaymanager kann nicht fortfahren

Diese Nachricht zeigt an, daß die Funktion BLIOVLRES() aufgerufen wurde, jedoch der Overlay-Manager nicht genügend Speicher belegen kann, um weiterhin Overlays auszuführen.

Der mögliche Grund dieser Meldung ist in der Speicher- belegung nach Aufruf der Funktion BLIOVLSUS() zu suchen. Ist der Overlay-Manager aus dem Speicher entfernt, wird wahrscheinlich eine Routine aufgerufen, die Speicher belegt, ihn aber abschließend nicht wieder frei gibt. Das heißt, daß zu diesem Zeitpunkt dann zu wenig freier Speicher für die Wiederaufnahme der Overlay-Aktivitäten vorhanden ist.

1217: Overlayvektor bei Programmausführung verbogen

Diese Meldung zeigt an, daß der Overlaymanager einen ungültigen Overlayvektor entdeckt hat. Der wahrscheinlichste Grund dafür ist, daß dieser zur Laufzeit zerstört wurde, weil ein Zeiger ungültig war oder weil versucht wurde, ein Modul in den Overlay-Bereich zu laden, das nicht overlayfähig ist.

Ein anderer Grund könnte darin liegen, daß in einem Debugger ein Breakpoint in den Code nach einem Overlay-Aufruf gesetzt wurde und dadurch die Vektordaten gegen einen Debugger-Breakpoint ausgetauscht wurden.

Verwenden Sie den MURPHY-Befehl, wenn Sie Overlays debuggen. Damit können Sie identifizieren, ob ein Modul, das nicht in ein Overlay gelegt werden darf, für diese Meldung verantwortlich ist.

1218: Symbol <Symbolname> wurde nicht in das Programm gelinkt

Diese Meldung sagt aus, daß der angegebene Symbolname nicht in das .EXE-Hauptprogramm gelinkt wurde, aber eine Referenz von einer DLL erhielt. Diese Meldung erscheint typischerweise nur bei CA-Clipper-Programmen.

Stellen Sie sicher, daß eines der in das .EXE-Hauptprogramm gelinkten Module eine Referenz auf das angegebene Symbol besitzt. Der Code, der sich so verhält, muß nicht wirklich ausgeführt werden - er muß jedoch mit dazugelinkt werden, um sicherzustellen, daß die entsprechenden Laufzeit-Funktionen in der .EXE-Datei enthalten sind.

1219: BLXCLPnn.OBJ muß in das .EXE-Hauptprogramm gelinkt werden

Diese Meldung besagt, daß das Programm versuchte, auf eine CA-Clipper-DLL zuzugreifen, die entsprechende Datei BLXCLPnn.OBJ jedoch nicht in das .EXE-Hauptprogramm gelinkt worden ist. Diese Meldung tritt nur bei CA-Clipper-Programmen auf.

Fügen Sie den Befehl FILE BLXCLPnn, wobei nn der Version Ihres eingesetzten CA-Clippers entspricht, an das Ende der FILE-Anweisungen für das .EXE-Hauptprogramm in die Linkskriptdatei ein.

Swap-Laufzeitfehlermeldungen (1220)

Fehler 1220 ist die Fehlermeldung der Sap- und Run-Funktion

1220: Swap-Fehler [major/minor] (Textmeldung)

Diese Nachricht weist auf einen schwerwiegenden Fehler im Blinker-Swapsystem hin. Die Meldung enthält eine Hauptfehlernummer, eine zugehörige untergeordnete Fehlernummer [Haupt-/Unternummer] und einen Text mit der Bezeichnung des Fehlertyps des aufgetretenen Fehlers.

Normalerweise werden solche Fehler durch die der Swap-Funktion zuzuordnenden Fehlerfunktionen SWPERRMAJ() und SWPERRMIN() gemeldet. Doch wenn das Programm im Speicher nicht wiederhergestellt werden kann, wird ein Fehler 1220 ausgegeben und das Programm gibt die Steuerung an DOS zurück.

Die folgenden übergeordneten Fehlercodes besitzen folgende Bedeutung und Textnachrichten:

0 Kein Fehler

1 Diskette/Platte voll

2* Datei nicht gefunden

3* Schreib-/Lesefehler auf Platte

4 Fehler bei Verzeichniswechsel

5 Fehler bei Erstellung einer temporären Datei

6 Fehler bei Laufwerkswechsel

7* Fehler bei EMS-Funktion

8* Fehler bei XMS-Funktion

9 SHELL-Fehler

10 Handle-Tabelle zu groß zum Speichern

11+ Fehler bei oberer Speichergrenze

12+ MCB-Kette defekt

13 Zu viele MCBs

14* Fehler bei DOS-Speicherfunktionsaufruf

15+ Segment-Zuweisungsfehler

16 Kann COMSPEC in Umgebung nicht finden

254+ Interner SWAP-Fehler

255+ Interner SWAP-Fehler

Mit einem Pluszeichen (+) gekennzeichnete Fehler sind immer schwerwiegend und erzeugen einen Fehler 1220. Mit einem Stern (*) gekennzeichnete Fehler können eine solche 1220 Fehlermeldung zur Folge haben. Sie können aber auch leichterer Natur sein, je nach dem, an welchem Punkt des Swapprozesses der Fehler aufgetretenen ist.

Fehler, die weder mit (+) noch mit (*) gekennzeichnet sind, sind nicht schwerwiegend (und haben daher auch keinen Fehler 1220 zur Folge), jedoch wird der untergeordnete Prozeß nicht ausgeführt. Diese

Fehler können über die Funktionen SWPERRMAJ() und SWPERRMIN() abgefangen werden (siehe Online-Hilfe über Blinker Swap Funktionen).

Es folgt nun die Erläuterung der einzelnen Hauptfehler:

0 - Kein Fehler

Der Wert Null für einen Hauptfehler zeigt an, daß die Swap-Funktion in der Lage war, den Befehlsinterpreter und das Programm-Abbild erfolgreich wiederherzustellen.

Normalerweise ist der Code für den untergeordneten Fehler ebenfalls Null. In dem Fall, daß die Swap-Funktion ein speicherresidentes Programm aus dem Speicher entfernen mußte, ist der Code für den untergeordneten Fehler für rein informative Zwecke gleich 1.

Wenn sich die Größe des DOS-Speichers während der Auslagerung vergrößert hat (durch Ausführen eines Programms wie VIDRAM von Quarterdeck), dann wird als Code für den untergeordneten Fehler eine 2 zurückgegeben. Benutzer sollten von der Ausführung solcher Programme während des Swappens UNBEDINGT Abstand nehmen. Beachten Sie bitte, daß eine Verringerung des DOS-Speichers durch die Verwendung eines solchen Dienstprogramms zu einem schwerwiegenden Fehler führt (Hauptcode 11).

1 - Diskette/Platte voll

Die Diskette/Festplatte, auf der die Swap-Funktion das Programm-Abbild speichern wollte, ist voll.

Der untergeordnete Fehlercode entspricht dem zugehörigen DOS-Fehlercode.

2 – Datei nicht gefunden

DOS hat den Fehler "Datei nicht gefunden" erzeugt beim Versuch der Swap-Funktion, das Programm-Abbild von der Swapdatei wiederherzustellen. Es kann sein, daß die Swapdatei gelöscht wurde.

Der untergeordnete Fehlercode entspricht dem zugehörigen DOS-Fehlercode.

3 - Schreib-/Lesefehler auf Platte

Beim Lesen oder Schreiben einer physischen Datei mit dem Programm-Abbild durch die Swap-Funktion trat in DOS unerwartet ein Fehler auf.

Der untergeordnete Code entspricht dem DOS-Fehlercode.

4 - Fehler bei Verzeichniswechsel

Beim Wechsel zu dem als Parameter übergebenen Verzeichnis durch die Swap-Funktion oder beim Wiederherstellen des aktuellen Verzeichnisses nach dem Swap-Vorgang ist ein Fehler aufgetreten.

Der untergeordnete Code entspricht dem zugehörigen DOS-Fehlercode.

5 - Fehler bei Erstellung einer temporären Datei

Beim Versuch der Erstellung einer temporären Datei durch die Swap-Funktion zum Zwecke der Wiederherstellung des Programm-Abbildes ist in DOS ein Fehler aufgetreten. Überprüfen Sie, ob das für temporäre Dateien angegebene Verzeichnis gültig ist.

Der untergeordnete Code entspricht dem zugehörigen DOS-Fehlercode.

6 - Fehler bei Laufwerkswechsel

Beim Wechsel zu dem als Parameter übergebenen Laufwerk durch die Swap-Funktion oder beim Wiederherstellen des aktuellen Laufwerks nach dem Swap-Vorgang ist ein Fehler aufgetreten.

Der untergeordnete Code entspricht dem zugehörigen DOS-Fehlercode.

7 - Fehler bei EMS-Funktion

Während einer EMS-Operation ist ein unerwarteter Fehler aufgetreten.

Der untergeordnete Code enthält den fehlgeschlagenen EMS-Funktionsaufruf.

8 - Fehler bei XMS-Funktion

Während einer XMS-Operation ist ein unerwarteter Fehler aufgetreten.

Der untergeordnete Code enthält den fehlgeschlagenen XMS-Funktionsaufruf.

9 - SHELL-Fehler

Die Swap-Funktion war nicht in der Lage, den Kommandoprozessor zu starten. Überprüfen Sie, ob die Umgebungsvariable COMSPEC auf den Kommandointerpreter zeigt.

Der untergeordnete Code entspricht dem zugehörigen DOS-Fehlercode.

10 - Handle-Tabelle zu groß zum Speichern

Die Swap-Funktion ist nur in der Lage, eine erweiterte Handle-Tabelle mit maximal 255 Handles abzuspeichern.

Der untergeordnete Fehlercode enthält die Anzahl der Handles in der aktuellen Handle-Tabelle.

11 – Fehler bei oberer Speichergrenze

Die obere Begrenzung des DOS-Speichers befindet sich nach der Auslagerung an einer niedrigeren Adresse als vorher. Arbeiten Sie während des Auslagerns eines Programms auf keinen Fall mit solchen Programmen wie dem Dienstprogramm VIDRAM von Quarterdeck.

Der untergeordnete Fehlercode enthält die Segmentadresse des oberen Speicherendes.

12 - MCB-Kette defekt

Die Swap-Funktion ist auf eine beschädigte DOS-MCB-Kette gestoßen (MCB = memory control block).

Der untergeordnete Fehlercode enthält die Segmentadresse des beschädigten MCB.

13 - Zu viele MCBs

Die Swap-Funktion kann nur solche Programme speichern, die maximal 128 DOS-MCBs (MCB = memory control block) enthalten, was in den meisten Fällen mehr als ausreichend ist.

Der untergeordnete Fehlercode enthält die Anzahl der MCBs.

14 - Fehler bei DOS-Speicherfunktionsaufruf

DOS hat einen Fehler bei einem speicherbezogenen Funktionsaufruf zurückgegeben. Nehmen Sie bitte Kontakt mit dem technischen Support auf und legen Sie diesem alle Einzelheiten zu der Situation vor, durch die dieser Fehler hervorgerufen wurde.

Der untergeordnete Code entspricht dem zugehörigen DOS-Fehlercode.

15 - Segment-Zuweisungsfehler

Um ein Programm-Abbild korrekt wiederherstellen zu können, muß die Swap-Funktion in der Lage sein, die gleichen Speicherblöcke belegen zu können wie vor der Auslagerung. Diese Nachricht teilt Ihnen mit,

daß DOS einen Block mit einer anderen Adresse zurückgegeben hat, als dies von der Swap-Funktion erwartet wurde. Daher kann das Programm nicht korrekt wiederhergestellt werden.

Der untergeordnete Fehlercode enthält die von DOS zurückgegebene Segmentadresse.

16 - Kann COMSPEC in Umgebung nicht finden

Die Swap-Funktion muß in der Lage sein, den Kommandointerpreter (normalerweise die Datei COMMAND.COM) zu finden, um einen untergeordneten Prozeß ausführen zu können. Dieser Fehler zeigt an, daß die Umgebungsvariable COMSPEC nicht gefunden werden konnte. Über diese Variable wird das Verzeichnis des Kommandointerpreters angegeben. Überprüfen Sie, ob die COMSPEC-Einstellung in Ihrer Umgebung auf ein gültiges Verzeichnis verweist, in dem sich das erwartete Programm befindet.

254 - Interner SWAP-Fehler

Nehmen Sie bitte Kontakt zu dem technischen Support auf und geben Sie diesem alle Einzelheiten der Situation an, durch die der Fehler hervorgerufen wurde.

255 - Interner SWAP-Fehler

Nehmen Sie bitte Kontakt zu dem technischen Support auf und geben Sie diesem alle Einzelheiten der Situation an, durch die der Fehler hervorgerufen wurde.

DOS-Protected Mode-Laufzeitfehler (13xx)

Fehler 1301 bis 1399 sind Protected Mode Laufzeitmeldungen

1301: kann Modul <Modulname> nicht finden oder laden

Diese Nachricht gibt an, daß der DOS-Extender nicht in der Lage war, das angegebene Modul, d.h. die .EXE oder .DLL Datei, im aktuellen Verzeichnis oder in dem mit PATH angegebenen Verzeichnis zu finden. Oder das Modul konnte gefunden werden, aber es gab ein Fehler beim Laden.

Wenn eine Datei mit dem richtigen Namen existiert, ist die wahrscheinlichste Ursache dieses Fehlers, daß die Datei überschrieben oder beschädigt wurde, möglicherweise wegen eines abgebrochenen Linkprozesses bei der Erzeugung der Datei. Versuchen Sie bitte, die betroffene Datei erneut zu installieren oder zu erzeugen.

Es kann ebenfalls sein, daß der derzeitige Benutzer zwar die Netzwerkrechte zum Öffnen dieser Datei besitzt, aber nicht die Rechte, diese zu laden und auszuführen.

1302: Fehler bei DPMI-Funktionsaufruf nnnnh, Fehlercode nnh

Diese Nachricht zeigt an, daß der angegebene Fehlercode über den Aufruf der spezifizierten DPMI-Funktion hervorgerufen wurde. Weitere Informationen zu der Bedeutung des Funktionsaufrufs sowie des Fehlercodes finden Sie in Ihrer DPMI-Dokumentation.

1303: DOS-Funktionsaufruf nnh wird nicht unterstützt

Diese Nachricht gibt an, daß das Programm versucht hat, eine bestimmte DOS-Funktion aufzurufen, die aber durch den DOS-Extender nicht unterstützt wird. Entspricht dies nicht dem erwarteten Verhalten, dann sollten Sie mit dem technischen Support von Blink, Inc Kontakt aufnehmen und diesem Ihre Blinker-Seriennummer sowie die Einzelheiten des Problems mitteilen.

1304: Kann XMS-Host mit der aktuellen Konfiguration des Speichermanagers nicht verwenden

Diese Meldung gibt an, daß der DOS-Extender nicht in den Protected Mode umschalten kann, da die Konfiguration der 386-Speichermanager-Software dies verhindert.

Wenn ein 386-Speichermanager vorhanden ist, wird dem DOS-Extender nicht erlaubt, seine eigenen Routinen zum Umschalten des Prozessor-Modus zu verwenden, um in den Protected Mode zu gehen bzw. diesen wieder zu verlassen. Stattdessen muß er entweder die Modus-Umschaltfunktionen von VCPI oder DPMI verwenden, die die Speichermanagement-Software zur Verfügung stellt.

Diese Meldung weist darauf hin, daß ein 386-er Speichermanager (oder etwas Entsprechendes) auf dem Rechner läuft, aber für den Extender keine DPMI- oder VCPI-Dienstleistungen zur Verfügung stehen (oder der Extender wurde durch Befehle zur Linkzeit oder durch die BLINKER-Umgebungsvariable so konfiguriert, daß er diese Dienstleistungsfunktionen nicht verwenden soll).

Lösungen:

Wenn möglich, konfigurieren Sie den Speicher-Manager so, daß er VCPI-Dienste zur Verfügung stellt (Hinweis: durch Einschalten von EMS wird normalerweise auch VCPI eingeschaltet).

Wenn möglich, konfigurieren Sie den Speicher-Manager so, daß er DPMI-Dienste zur Verfügung stellt.Vergewissern Sie sich, daß die Nutzung von VCPI oder DPMI durch den Extender nicht deaktiviert wurden durch Befehle zur Linkzeit (z.B. BLINKER HOST VCPI OFF) oder durch Laufzeit-Einstellungen

in der Umgebungsvariablen BLINKER.

Sehr alte Speicher-Manager stellen eventuell keine VCPI-Dienste zur Verfügung. Solche Speicher-Manager müssen eventuell entfernt werden, bevor der Blinker-Extender arbeiten kann.

1305: Grenzwert für größten Selektor überschritten

Diese Nachricht teilt Ihnen mit, daß die maximale Anzahl der für ein Programm erlaubten Selektoren (ungefähr 8000) überschritten wurde. Das ist gewöhnlich bei einem Anwendungsprogramm mit einer großen Anzahl kleiner Segmente der Fall.

Das Problem kann gelöst werden, indem das Programm so verändert wird, daß größere Segmente angelegt werden. Diese werden dann innerhalb des Programms weiter unterteilt, in der gleichen Weise, wie die C-Standardbibliotheksfunktionen malloc() und free() funktionieren.

1306: ungültiger Selektorwert

Diese Nachricht gibt an, daß an eine interne Extenderfunktion eine ungültige Selektornummer übergeben wurde. Dies dürfte nicht in normalen Operationen auftreten.

1307: kann Segment im Modul <Modul> nicht laden

Diese Nachricht zeigt an, daß das Ladeprogramm des DOS-Extenders nicht in der Lage war, ein Segment zu laden, weil es nicht vorhanden war und daher eine Ausnahme ausgelöst wurde. Dies entspricht dem Fall, daß ein Overlay nicht geladen werden kann und zeigt normalerweise an, daß die .EXE-Datei beschädigt oder überschrieben wurde. Versuchen Sie bitte, die fehlerbehaftete .EXE-Datei erneut zu installieren oder zu erzeugen.

1308: Abbruch der Ausführung durch Anwender

Diese Nachricht teilt Ihnen mit, daß der Benutzer die Tastenkombinationen Ctrl-C oder Ctrl-Break gedrückt hat und daß das DOS-Extender-Programm keinen Control-Break-Handler installiert hat, so daß die Ausführung beendet wurde.

1309: Kein Speicher mehr verfügbar

Diese Nachricht gibt an, daß die durch das virtuelle Speichersystem erzeugte temporäre Swap-Datei zu groß für die Diskette/Festplatte ist, auf der sie erstellt wurde, oder daß die Begrenzung des virtuellen Speichersystems von 64 MB erreicht wurde.

Überprüfen Sie, ob das Programm nicht eventuell immer wieder neuen Speicher belegt, ohne nicht benötigten wieder freizugeben (z.B. in einer Schleife). Wenn das nicht der Fall ist, dann sollten Sie auf dem temporären Laufwerk mehr Speicher zur Verfügung stellen oder das Laufwerk, auf das der virtuelle Speicher ausgelagert wird, wechseln.

1310: Dieses Programm erfordert mindestens einen 286-Prozessor

Diese Nachricht zeigt an, daß versucht wurde, das DOS-Extender-Programm auf einem Rechner auszuführen, der keinen 80286-er Prozessor (oder höher) besitzt.

Protected Mode wird nur auf 286-er und späteren Prozessoren unterstützt.

1311: Dieses Programm erfordert mindestens DOS 3.0

Diese Nachricht zeigt an, daß versucht wurde, das DOS-Extender-Programm unter einer DOS-Version auszuführen, die älter als 3.0 ist. Dies wird nicht unterstützt.

1312: Nicht genügend Extended Memory verfügbar, um das Programm zu starten

Diese Nachricht gibt an, daß nicht genügend Erweiterungsspeicher zum Laden und Ausführen des Programms zur Verfügung steht. Der Speicherbedarf zum Ausführen eines Programms hängt normalerweise von der Größe des Programms ab. Es kann aber auch zur Linkzeit über den Befehl BLINKER EXECUTABLE EXTENDED nnnn ein Minimum gesetzt worden sein.

1313: Ausnahmefehler nn : <(Exception-Typ)>, Code = nnnnh

Diese Nachricht teilt Ihnen mit, daß während der Ausführung des Programms eine Schutzverletzung des angegebenen Typs aufgetreten ist. Das weist auf einen Fehler im Anwendungsprogramm hin und ist kein DOS-Extender-Fehler. Die Nachricht besagt, daß das Anwendungsprogramm versucht hat, eine unerlaubte Operation durchzuführen.

Die Ausnahmetypen und eine kurze Beschreibung ihrer Ursachen sind weiter unten aufgeführt. Der numerische Code enthält häufig den Wert, durch den der Fehler hervorgerufen wurde, zum Beispiel den Wert eines unzulässigen Selektors, der in ein Segmentregister geladen wurde, oder einen Zeiger, der auf einen Punkt hinter der Begrenzung eines Segments verweist.

Dieser Fehlermeldung folgt die Ausgabe der Inhalte aller Prozessorregister und verschiedener anderer Informationen. Eine detailliertere Analyse dieser Informationen finden Sie in der Online-Hilfe im Abschnitt "Analyse von Allgemeinen Schutzverletzungen (GPFs)".

Exception 00: Division durch Null versucht

Diese Verletzung wird durch einen Versuch erzeugt, einen Wert durch Null zu dividieren. Sie wird normalerweise durch die Laufzeitbibliothek des Compilers abgefangen, wenn mit einer Hochsprache gearbeitet wird.

Exception 05: Fehler bei Überprüfung von Grenzwerten

Diese Verletzung wird durch eine BOUND-Anweisung erzeugt, wenn der zu testende Wert kleiner als die angegebene Untergrenze oder größer als die angegebene Obergrenze ist.

Exception 06: nicht definierter Opcode

Diese Verletzung wird durch einen Versuch erzeugt, einen Opcode (Prozessorbefehl) auszuführen, der nicht für den Prozessor definiert ist, unter dem das Programm läuft.

Exception 07: kein Math.-Koprozessor vorhanden

Diese Verletzung wird durch einen Versuch ausgelöst, eine Gleitkomma-Anweisung auszuführen, obwohl der Prozessor gar keinen mathematischen Coprozessor besitzt.

Exception 08: doppelter Ausnahmefehler (Exception-Fehler)

Diese Verletzung wird generiert, wenn innerhalb einer Anweisung mehrere Verletzungen ausgelöst wurden, oder in einem Fehler-Handler eine Verletzung auftritt.

Exception 0a: ungültiges Segment für Taskwechselstatus

Diese Verletzung wird durch einen Versuch erzeugt, zu einem Segment mit einem fehlerhaften Taskstatus umzuschalten.

Exception 0c: Stack-Überlauf / -Unterlauf

Dieser Fehler wird durch die Nichtbeachtung einer Begrenzung in einer beliebigen Operation hervorgerufen, die auf das Stacksegment verweist. Dies umfaßt stack-orientierte Anweisungen wie PUSH und POP ebenso wie andere Speicherverweise, die den Stack implizit nutzen, wie z.B. MOV AX,[BP].

Exception 0d: Allgemeine Schutzverletzung

Alle anderen, bisher nicht genannten Schutzverletzungen verursachen eine allgemeine Schutzverletzung. Dies umfaßt die Überschreitung von Segmentgrenzen, den Versuch, ungültige Selektoren in Segmentregister zu laden, oder auch den Versuch, Daten auszuführen oder in Code hineinzuschreiben.

Dies ist der häufigste Typ von Protected Mode Laufzeitfehler, und er wird genauer beschrieben in der Online-Hilfe im Abschnitt "Analyse von Allgemeinen Schutzverletzungen (GPFs)".

Exception 0e: Seitenfehler

Diese Verletzung wird erzeugt beim Versuch, ein Segment an einer ungültigen Speicheradresse zu laden.

1314: kann Datei <Dateiname> nicht erzeugen

Diese Nachricht zeigt an, daß das virtuelle Speichersystem des DOS-Extenders nicht in der Lage war, die angegebene temporäre Swap-Datei zu erzeugen.

Die wahrscheinlichsten Ursachen für diese Fehlermeldung sind, daß entweder bereits eine Datei desselben Namens existiert, die schreibgeschützt ist, oder daß keine Erzeugungsrechte für das Ziellaufwerk oder Verzeichnis gewährt wurden, oder daß das Ziellaufwerk voll ist.

1315: konventioneller Speicher nicht ausreichend, um Programm zu starten

Diese Meldung zeigt an, daß ein Dual Mode-Programm versucht hat, im Real Mode zu starten, aber dabei entdeckt hat, daß nicht genügend konventioneller (DOS)-Speicher vorhanden ist, um den Ladevorgang abzuschließen.

Stellen Sie mehr konventionellen Speicher zur Verfügung, indem Sie z.B. residente Programme entfernen. Danach starten Sie das Programm neu. Außerdem finden sie in der Online-Hilfe weitere Hinweise, wie man noch mehr Code eines Real Mode-Programms in den Overlay-Bereich verlagern kann.

1316: Platte voll für VM-Swap-Datei oder VM-Begrenzung überschritten

Diese Meldung weist darauf hin, daß die virtuelle Speicherverwaltung des DOS-Extenders von Blinker versucht, auf die Festplatte auszulagern (swap), und das Ziellaufwerk war entweder ungültig oder voll oder die maximale Größe für eine Swap-Dateien von 64MB wurde überschritten.

Die Voreinstellung für das Zielverzeichnis, in dem Swap-Dateien angelegt werden, wird in den Umgebungsvariablen TMP oder TEMP angegeben. Wenn keine von beiden Umgebungsvariablen gesetzt ist, wird das aktuelle Verzeichnis verwendet. Überprüfen Sie, ob auf ein gültiges Ziellaufwerk geschrieben wird, das auch genügend viel Platz für die temporäre VM-Swapdatei des DOS-Extenders hat.

1317: Extender-Heap voll

Diese Meldung weist darauf hin, daß der Blinker DOS-Extender-Heap völlig aufgebraucht ist. Sie sollte nomalerweise nicht erscheinen.

Sie bedeutet meistens, daß eine große Anzahl kleiner Memory-Zuweisungen direkt über das Extender-API gemacht wurde. Dies geschieht meistens, wenn ein temporärer Puffer wiederholt zugewiesen wurde, ohne daß er jedesmal freigegeben wird.

Überprüfen Sie das Programm auf endlose Rekursionen oder darauf, ob eine temporäre Memory-Zuweisung nicht freigegeben wird, die sich in einer mehrfach aufgerufenen Funktion befindet.

1318: Division durch Null versucht bei (nn) ssss:oooo

Diese Meldung zeigt an, daß der Code in .EXE Segment nn an der Adresse ssss:oooo eine Integer

Division mit einem Divisor von Null ausgeführt hat und ein "Divide by zero"-Handler nicht installiert war.

1319: kann Namen <Modul.Importname> im Modul <Modul> nicht importierten

Diese Meldung zeigt an, daß das 2. Modul in der Meldung den angegebenen Import-Namen angefordert hat, der Blinker DOS-Extender ihn jedoch in den geladenen DLLs nicht auflösen konnte.

Dieser Fehler tritt gewöhnlich auf, wenn eine DLL mit einer neueren oder einer älteren DLL überschrieben wurde , welche keine Exportdefinition für den angegebenen Funktionsnamen enthält,.

Suchen Sie die DLL, die den importierten Namen enthalten sollte, prüfen Sie, daß es sich um die richtige Version handelt und verwenden Sie einen Import-Bibliotheksmanager wie BLILIB.EXE (siehe Online-Hilfe über ‘BLILIB Import Bibliotheksmanager’), um zu prüfen, daß sie den benötigten Namen exportiert.

1320: kann Ordnungsnummer <Modul.Nummer> im Modul <Modul> nicht importieren.

Diese Meldung zeigt an, daß das 2. Modul in der Meldung die angegebene Import-Nummer und das zugehörige Modul angefordert hat, der Blinker DOS-Extender sie jedoch in den geladenen DLLs nicht auflösen konnte.

Dieser Fehler tritt gewöhnlich auf, wenn eine DLL mit einer neueren oder einer älteren DLL überschrieben wurde, welche keine Exportdefinition für die angegebene Nummer der Funktion enthält,.

Suchen Sie die DLL, welche die importierte Ordnungsnummer enthalten sollte, und prüfen Sie, ob es sich um die richtige Version handelt.

1321: <Dateiname> muß mit Blinker 5.x komprimiert sein

Diese Meldung bedeutet, daß zwar die DOS Extended .EXE Datei mit Blinker 5.x komprimiert wurde, aber die angegebene .DLL Datei mit einer vorherigen Version von Blinker komprimiert wurde und nun nicht entpackt werden kann.

Erzeugen Sie die angegebene .DLL Datei durch Linken mit Blinker 5.x neu.

Andere häufige Fehler

DOS open error 4

DOS open error 5

Parameterwert außerhalb des zulässigen Bereichs

Sharing violation

Das System kann die Datei USER nicht finden

USER.DLL kann nicht geladen werden

Andere häufige Fehler

DOS open error 4

Das ist ein DOS-Fehler, der angibt, daß keine weiteren Handles zum Öffnen einer weiteren Datei mehr frei sind. Erhöhen Sie die Anzahl der Datei-Handles an jeder der drei folgenden Stellen:

- In der Arbeitsstation - FILES=nn in CONFIG.SYS
- Im Netzwerk - (SHELL.CFG für Novell)
- Im Programm - einige C-Compiler begrenzen die Anzahl der Datei-Handles in der Startroutine.

Denken Sie daran, daß Blinker pro Overlay ein Datei-Handle benötigt: eins für ein internes Overlay oder bei der Verwendung von externen Overlays eins pro SECTION INTO-Befehl. Ebenso benötigt CA-Clipper 5.x eine entsprechende Anzahl von Datei-Handles für sein dynamisches Paging-System.

Andere häufige Fehler

DOS open error 5

Insbesondere in DOS 4.0 und 5.0 ,wo SHARE automatisch geladen wird, belegt SHARE Speicher für eine Tabelle mit einer festen Größe, in der für jede geöffnete Datei der Dateinamen und der vollständige Pfad und andere Dateiinformationen abgelegt wird. Dieser Fehler zeigt häufig an, daß in dieser Tabelle kein Platz zum Öffnen einer weiteren Datei mehr ist.

In diesem Fall müssen Sie den Befehl DOS SHARE mit einem größeren Wert für den Parameter /F;nnnn ausführen, um mehr Platz zu reservieren.

Andere häufige Fehler

Parameterwert außerhalb des zulässigen Bereichs

Diese Fehlermeldung wird von COMMAND.COM ausgegeben, wenn der Swapper ein anderes Programm startet und die DOS Umgebung der Maschine sehr klein ist. SWPRUNCMD() bestimmt die augenblickliche Größe der Umgebung, addiert die zusätzlich benötigte Umgebungsgröße (falls erforderlich) und fügt einen /E:nnn Parameter zum COMMAND.COM. Falls der berechnete /E:nnn Wert kleiner als der Standardwert von COMMAND.COM ist, gibt COMMAND.COM diese Fehlermeldung aus.

Um dieses Problem zu umgehen, ignorieren Sie die Meldung einfach (es passiert nichts), oder vergrößern Sie den Umgebungsbereich. Dies kann erreicht werden durch zusätzliche Einstellungen für die Hauptumgebung in der AUTOEXEC.BAT, oder durch einen zusätzlichen Aufruf der Funktion SWPADDENV(128) vor einem Swap, wodurch die Umgebung auf eine Größe oberhalb des unteren Grenzwertes von COMMAND.COM erweitert wird.

Andere häufige Fehler

Sharing Violation

In verschiedenen Netzwerken einschließlich 3COM und LANTASTIC treten Sharing Violation-Fehler bei internen Overlays auf, wenn eine zweite Kopie des Programmes geladen ist. Das scheint daher zu kommen, daß das Netzwerk exklusiven Schreibzugriff auf die Programmdatei benötigt, wenn es die Datei lädt.

Um dieses Problem zu umgehen, können Sie mit Hilfe des Blinker-Befehls READONLY (oder mit dem DOS-Dienstprogramm ATTRIB.EXE) der .EXE-Datei das Attribut NUR LESEN festlegen. Sie können auch in der Skriptdatei über den Blinker-Befehl SECTION INTO zu Beginn des BEGINAREA ein externes Overlay angeben.

Andere häufige Fehler

Das System kann die Datei USER nicht finden

Der Befehlsprozessor von Windows NT 3.51 und später versucht immer, DOS Extended Programme, die .DLLs verwenden, als OS/2 Programme zu laden. Dies funktioniert natürlich nicht und führt zu einer Fehlermeldung, die in etwa besagt : ‘Das System kann die Datei USER nicht finden’. Um Windows NT zu zwingen, das Programm als ein DOS Programm zu starten, geben am Anfang der Befehlszeile FORCEDOS ein, wodurch das Programm x:\WINNT\SYSTEM32\FORCEDOS.EXE gestartet wird.

Z.B.:

```
FORCEDOS TEST.EXE <parameters>
```

Die Datei BLINKER.COM, im Blinker enthalten, erzwingt, daß eine .EXE Datei mit gleichem Namen im selben Pfad immer als DOS Extended Programm geladen wird. Also kann man alternativ BLINKER.COM als TEST.COM in das gleiche Verzeichnis kopieren und explizit TEST.COM statt TEST.EXE starten. Z.B.:

```
TEST.COM <parameters>
```

Andere häufige Fehler

USER.DLL kann nicht geladen werden

Siehe den Eintrag für:

Das System kann die Datei USER nicht finden

Anhänge

VERBOSE Ausgabe

MAP-Ausgabe - ausführbare Real Mode-Dateien

Map-Ausgabe - DOS-Extender- und Windows-Programme

Literaturhinweise

VERBOSE Ausgabe

Die VERBOSE-Ausgabe ist oft hilfreich, wenn Fehler aufgespürt werden sollen.

Während der Verwendung von VERBOSE können Sie die Reihenfolge, in der Blinker die .OBJ- und die .LIB-Dateien abarbeitet, um die .EXE-Datei zu erstellen, genau mitverfolgen. Falls Blinker seine Arbeit nicht vollständig verrichten konnte, so ist der VERBOSE-Befehl ebenfalls hilfreich bei der Suche nach dem Modul, welches den Fehler ausgelöst hat, da in der Regel das zuletzt angezeigte Modul die Ursache ist.

Die Verwendung des Blinker-Befehls VERBOSE 1 (Namen der bearbeitete Dateien ausgeben) in einer Beispiel-Linkdatei ergab das folgende Ausgabe-Listing:

```
File : TEST.OBJ
File : CLIPPER.LIB
File : EXTEND.LIB
File : TERMINAL.LIB
File : DBFNTX.LIB
    .
    .
    .
File : TEST.OBJ
File : CLIPPER.LIB
File : EXTEND.LIB
File : TERMINAL.LIB
File : DBFNTX.LIB
```

Die Verwendung von VERBOSE 2 (Namen der bearbeiteten Dateien und Module ausgeben) ergab folgendes Listing, das auch die Grundlage für unsere darauffolgende Besprechung sein wird:

```
TEST.OBJ(TEST)
CLIPPER.LIB(CRT0)
CLIPPER.LIB(CRT0DAT)
CLIPPER.LIB(CRT0FP)
CLIPPER.LIB(CRT0MSG)
    .
```

```
    .
    .
EXTEND.LIB(GETE)
EXTEND.LIB(DISKSPAC)
EXTEND.LIB(_DIR)
EXTEND.LIB(COPYFILE)
    .
    .
    .
DBFNTX.LIB(DTX0)
DBFNTX.LIB(DTXDYN)
Resolving external references
Building map table
*****  Pass 2  *****
TEST.OBJ(TEST)
CLIPPER.LIB(CRT0)
CLIPPER.LIB(CRT0DAT)
CLIPPER.LIB(CRT0FP)
    .
    .
    .
EXTEND.LIB(_TBROWSE)
EXTEND.LIB(_TBRDB)
TERMINAL.LIB(BEEP)
TERMINAL.LIB(DISPIBM)
TERMINAL.LIB(KBDIBM)
DBFNTX.LIB(NET)
DBFNTX.LIB(DTX1)
DBFNTX.LIB(DTX0)
DBFNTX.LIB(DTXDYN)
Clearing output buffers
```

Wird Verbose ohne Parameter angegeben, dann wird VERBOSE 2 verwendet.

Schritt 1 - Listing

Der erste Teil der VERBOSE-Liste gibt alle Module aus, die durch Blinker bearbeitet werden, zu dem Zeitpunkt zu dem Sie entdeckt wurden. Die Bezeichnung ‘Modul’ bezieht sich generell auf eine Objekt-Datei, wobei diese Objekt-Datei durch das FILE-Kommando angegeben sein kann oder innerhalb einer Library existiert und somit mit dem LIBRARY-Kommando angegeben werden kann.

Die Haupt-Objektdatei oder -Library wird hierbei zuerst angegeben, gefolgt von dem in Klammern angegebenen Modul-Namen. Diese Module müssen nicht zwingend in die .EXE-Datei aufgenommen werden, werden aber dennoch, durch die Linkdatei gesteuert, bearbeitet.

Typischerweise ist der Modulname der Basisname der originalen Quelle, ohne jedoch den Laufwerksbezeicher, den Pfad und die Erweiterung zu beinhalten.

Modulnamen, die während der ersten Phase angezeigt werden, lassen sich wie folgt aufschlüsseln:

- alle Module, die durch die FILE-Anweisung spezifiziert sind
- alle Module einer Library, die zur Auflösung von unbekannten Symbolen benötigt werden.
- alle Module einer durch SEARCH angegebenen Library.

Schritt 2 - Listing

Während der zweiten Phase, zu der Blinker genau weiß, welche Module für das resultierende .EXE benötigt werden, werden auch nur noch die tatsächlich verwendeten Module angezeigt.

Diese Liste wird oft mit der Liste der doppelten Module

(erzeugt durch Angabe von BLINKER MESSAGE DUPLICATE) verwendet, um die Module der .EXE-Datei zu ermitteln, die während der Linkzeit auch doppelt angegeben wurden.

MAP-Ausgabe - ausführbare Real Mode-Dateien

Der Standard-Blinker MAP-Befehl produziert eine Liste der .EXE-Datei wie sie unten angegeben ist. Diese Liste wurde mittels der Option ‘S’ des MAP-Befehls erstellt.

```
Start     Stop      Length       Name          Class
00000h    02F48h    02F49h    _TEXT         CODE
02F4Ah    02F4Ah    00000h    SORTBLOC_TEXT CODE
02F4Ah    03213h    002CAh       ASUPPORT_TEXT CODE
03214h    035C0h    003ADh    FILESYS_TEXT  CODE
             .                    .
             .                    .
41E94h    41E94h    00000h    _SYMP         DATA
41E94h    41E94h    00000h    _SYMPE        DATA
41E94h    41EA1h    0000Eh    CDATA         DATA
41EA2h    41EA2h    00000h    XIFB          DATA
             .                    .
425B0h    425EBh    0003Ch    C_COMMON      BSS
425F0h    435EFh    01000h    STACK         STACK

Origin       Group
3C2A:0000    DGROUP

Program entry point at 3710:0399
```

Dies ist ein Beispiel mit verschiedenen Fragmenten einer durch den MAP-Befehl generierten Ausgabe.

Der Name und die Klasse jedes Segments innerhalb der .EXE-Datei ist unter der korrespondierenden Spaltenüberschrift aufgeführt. Diese Segment-Namen und Klassen wurden durch den jeweiligen Compiler oder durch den Programmierer der resultierenden Objekt-Datei zugewiesen.

Die Spalte mit der Überschrift START weist die Anfangsadresse jedes Segments innerhalb des Speicherabbilds auf. Die STOP-Spalte die dazugehörige End-Adresse. Die Länge ergibt sich einfach aus der Subtraktion der Endadresse von der Startadresse. Der Buchstabe ‘h’ bedeutet übrigens, daß die Zahlen als Hexadezimalzahlen zu lesen sind.

Da diese Adressen verschiebbar sind, muß die Lade-Adresse zu der MAP-Adresse hinzuaddiert werden, um die aktuelle Laufzeit-Adresse zu ermitteln.

Adreßliste der Overlay-Bereiche

In der Liste (Map) werden in einem gesonderten Abschnitt auch die Overlays des Programms (wenn vorhanden) aufgeführt. Darauf folgt die Segmentliste. Für jedes Overlay werden folgende Angaben gemacht: die (vom Blinker verwendete) Overlay-Nummer, die Anzahl der im Overlay enthaltenen Public-Symbole (diese entspricht normalerweise der Anzahl der Funktionen innerhalb eines Overlay), die Größe des Overlay in Bytes, der Name des in dem Overlay abgelegten Segments und der Name des Quellmoduls des Segments. Der Liste der Overlays folgt eine Gesamtzahl mit der Gesamtgröße aller Overlays innerhalb des Programms.

```
Section      Pubs   Length    Name      Module
0001H     1      250       MJINIT    MJINIT
0002H     1      68        DISFRG    MJINIT
0003H     1      42        DISPAK    MJINIT
```

```
   .   .   .      .   .   .
024DH     1      83         MJCOPY     MJCOPY
024EH     7      559        SWPMEM     CLSWPMEM
024FH     1      12817      $$$SWPSYS  SWPRUN

Total            425610
```

Diese Gesamtzahl entspricht der optimalen Größe für den Overlay-Cache. Wenn der Cache auf diese Größe gesetzt wird, können alle Overlays nebeneinander in dem Cache gespeichert werden. Wesentliche Geschwindigkeitsvorteile werden jedoch auch noch erreicht, wenn der Cache kleiner ist.

Hinweis: Blinker belegt in keinem Fall für den Cache mehr Speicherplatz als mit dieser Größe angegeben ist, auch wenn er andere Anweisungen erhalten hat.

Adreßliste der Public-Symbole

Die Option ‘A’ des MAP-Befehls listet alle Public-Symbole in ihrer Reihenfolge auf:

```
Address             Publics by Value
0000:0000     Abs   __START_SYM
0000:9876     Abs   __FLTUSED
0000:9876     Abs   __ACRTUSED
0000:9876     Abs   __ACRTMSG
      .       .     .
0001:0000     Res   TEST
00BE:0006     Res   PROD_MENU
00E7:000A     Res   COMP_MEN
      .       .     .
3E82:0000     Ovl   __APPINIT
3E83:0000     Ovl   ERRORSYS
3E86:0000     Ovl   __SETCENTU
3E87:0000     Ovl   __SETHELPK
```

Absolute Symbole, markiert durch ‘Abs’, sind typische Public-Konstanten, die die ganze Laufzeit der Applikation über im Speicher vorhanden sind, ebenso wie der BIOS Datenbereich oder der Bildschirmpuffer.

Residente Symbole, markiert durch ‘Res’ befinden sich immer in der Root des Programms.

Overlay-Symbole, markiert durch ‘Ovl’ werden ins Overlay gelegt.

Map-Ausgabe - DOS-Extender- und Windows-Programme

Bei der Erzeugung eines DOS-Extender- oder Windows-Programms erstellt Blinker eine Liste (Map) zu der ausführbaren Datei ähnlich der nachfolgend dargestellten Liste: Bei Angabe der Option ‘S’ wird die gleiche Map-Datei erzeugt.

```
 Start    Length Name             Class  Group
 0001:0000 0002BH   T                CODE
 0001:002C 03157H   _TEXT            CODE
 0001:3183 014BFH   SCREEN_DRIVER_TEXT  CODE
 0001:4642 0098AH   EXTEND_TEXT         CODE
   .   .   .
   .   .   .
 0006:2BE6 0000EH   CDATA            DATA   DGROUP
 0006:2BF4 00000H   XIFB             DATA   DGROUP
 0006:2BF4 00000H   XIF              DATA   DGROUP
 0006:2BF4 00000H   XIFE             DATA   DGROUP
   .   .   .
 0006:2F80 00000H   XOE              BSS    DGROUP
 0006:2F80 00000H   _STACK           STACK  DGROUP
 0006:2F80 01400H   STACK            STACK  DGROUP

 Origin   Group
 0003:0000 SYMGROUP
 0005:0000 DMTGROUP
 0006:0000 DGROUP

 Program entry point at 0001:036A
```

Dies ist ein Beispiel mit verschiedenen Teilen der Ausgabe, die während des Linkens eines DOS-Extender- oder Windows-Programms erstellt wird, wenn die Standardoption MAP angegeben wurde. Der Listenaufbau entspricht dem der Real Mode-Liste mit der Ausnahme, daß alle Segmente ab 1 beginnend durchgehend numeriert und die Segmente viel größer sind, da es sich um zusammengesetzte Segmente handelt.

Die Segmentnummern sind die Nummern, auf die verwiesen wird, wenn durch ein DOS-Extender-Programm eine Schutzverletzung hervorgerufen wurde.

Adreßliste der Public-Symbole

Die ‘A’-Option des MAP-Befehle listet alle Public-Symbole in der Reihenfolge ihrer Adressen auf:

```
  Address Publics by Value
 0000:0000  Abs  __START_SYM
 0000:9876  Abs  __FLTUSED
 0000:9876  Abs  __ACRTMSG
 0000:9876  Abs  __ACRTUSED
   .   .   .
 0001:01C8  Res  DOSALLOCREALSEG
 0001:0200  Res  DOSCREATEDSALIAS
 0001:0220  Res  DOSFREESEG
   .   .   .
 0006:2F0C  Res  _EDATA
 0006:2F0C  Res  __EDATA
 0006:2F80  Res  __END
 0006:2F80  Res  _END
```

Wie bei der Auflistung der Real Mode-Public-Symbole werden die absoluten Symbole durch ein ‘Abs’ und die residenten Symbole durch ein ‘Res’ gekennzeichnet.

Literaturhinweise

Intel 80286 and 80287 Programmers Reference Manual.
Intel Corporation.
Santa Clara, CA.
ISBN 1-55512-055-5

Extended DOS (2. Auflage). Ray Duncan
Addison Wesley Publishing Company, Inc.
Reading, MA.
ISBN 0-201-56798-9

Undocumented DOS. Andrew Schulman (ed).
Addison Wesley Publishing Company, Inc.
Reading, MA.
ISBN 0-201-57064-5

DOS Protected Mode Interface Specification 1.0
Intel Corporation.
Santa Clara, CA.
Intel Order No.240977-001

An Introduction to the DOS Protected Mode Interface.
Intel Corporation.
Santa Clara, CA.
Intel Order No.240787-003